ANIMATING PHYSICAL PHENOMENA WITH EMBEDDED SURFACE MESHES

A Thesis
Presented to
The Academic Faculty

by

Chris Wojtan

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology
December 2010

ANIMATING PHYSICAL PHENOMENA WITH EMBEDDED SURFACE MESHES

Approved by:

Professor Greg Turk, Advisor
School of Interactive Computing
Georgia Institute of Technology

Professor Irfan Essa
School of Interactive Computing
Georgia Institute of Technology

Professor C. Karen Liu
School of Interactive Computing
Georgia Institute of Technology

Professor Jarek Rossignac
School of Interactive Computing
Georgia Institute of Technology

Professor Peter J. Mucha
Institute for Advanced Materials,
Nanoscience and Technology
Carolina Center for Interdisciplinary
Applied Mathematics
The University of North Carolina at Chapel Hill

Date Approved: 12 November 2010

To my parents,

Loretta and Chester Wojtan,

who always believed in me and supported me

in pusuit of my lofty goals.

PREFACE

When I was a child, I loved to draw. I would spend hours each day staring at objects in my room and sketching them, aiming to make each one of my drawings look more realistic than the one before. I remember how excited I was when I learned how to add a sense of depth to a human face by shading around its edges and adding specular sparkles in the eyes. Although it was physically impossible to perfectly capture every visible nuance with only a pencil and a sheet of paper, I thoroughly enjoyed the process of distilling the essence of a physical object into a series of lines, plotting this essence onto paper, and somehow creating a crude copy that was eerily similar to the original. Our brains seem to find certain visual features vastly more important than others — these dominant features are essential for reproducing the likeness of an image, while it appears as though other features may be casually ignored.

I find that we are performing a similar process in the field of computer animation. My goal is to efficiently recreate extremely detailed physical motions, like splattering jelly and splashing liquid, inside of a computer. Due to limited computational resources, we cannot afford to simulate every atom involved in a physical phenomenon; we have to distill each physical action down to its essence and extract only the most important details while casually replacing other subtleties with simplistic approximations. This dissertation simulates highly deformable materials like liquids and viscoelastic goo by devoting the majority of a computer's attention to the motion of visible surfaces, while only approximating many subtle details in the way each object moves and changes topology. These ideas result in highly efficient methods for animating natural phenomena without sacrificing many important visual details.

ACKNOWLEDGEMENTS

I first wish to thank my wife Olya, for encouraging me, for supporting me, and for convincing me that all of my hard work was worth the effort. You make every day a thrill to be alive.

I also want to thank my mother Loretta, my father Chester, and my sister Amy. You each helped shape me into who I am today, and I am extremely grateful for this gift.

I want to thank the members of my committee: Jarek Rossignac, Irfan Essa, Peter J. Mucha, Karen Liu, and my advisor Greg Turk. Each of you helped me become a stronger researcher in your own way, from encouraging me to widen my research horizons and increase my rigor, to helping me develop social skills like personality and ambition.

I want to thank each of my collaborators: Adam W. Bargteil, Jessica K. Hodgins, Markus Gross, Nils Thürey, Nipun Kwatra, Mark Carlson, Irfan Essa, Peter J. Mucha, and Greg Turk. I thoroughly enjoyed working with each of you, and I hope we can continue collaborations in the future.

I want to especially thank my friends and labmates who helped me keep my sanity during the intense SIGGRAPH deadline periods: Moshe Mahler, Thomas Oskam, Tobias Pfaff, Huamin Wang, Brooks Van Horn, Mark Carlson, Arya Irani, Josh Zaritzsky, Sooraj Bhat, Raffay Hamid, David Roberts, Sumit Jain, Yuting Ye, Brian Whited, Byungmoon Kim, Karthik Raveendran, Jie Tan, Jason Williams, Matt Flagg, Chris Twigg, Jernej Barbic, Alla Safonova, Eakta Jain, Preethi Bhat, James Hays, Jim McCann, Bernd Bickel, Sebastian Martin, Peter Kaufmann, Hao Li, Balint Miklos, Johannes Schmid, Marcel Germann, Mario Botsch, and many more. I am sorry

if I neglected to mention anyone!

I wish to particularly thank Keenan Crane, James F. O'Brien, Adam W. Bargteil, and Nils Thürey, who have shown me repeatedly that I can turn to them for both technical advice and personal wisdom.

Finally, I want to express my deepest gratitude to my advisor Greg Turk. Your humility and brilliance are known worldwide, and they are not the least bit overstated. I feel deeply indebted to you, and I wish to thank you for being such an incredible mentor. You personally built up a huge portion of the knowledge and confidence that I possess today, you encouraged my creativity, you acted as my safety net, and you trusted me. I feel extremely, intensely fortunate that I ended up with you as my advisor, and I can say with complete confidence that I would be nowhere near as successful as I am today without you to guide me through my PhD. It has been an honor to be your apprentice for the last six years, and I envy my colleagues who will continue to spend time in your presence well after I leave.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Accurate computational representations of highly deformable surfaces are indispensable in the fields of computer animation, medical simulation, computer vision, digital modeling, and computational physics. The focus of this dissertation is on the animation of physics-based phenomena with highly detailed deformable surfaces represented by triangle meshes.

We first present results from an algorithm that generates continuum mechanics animations with intricate surface features. This method combines a finite element method with a tetrahedral mesh generator and a high resolution surface mesh, and it is orders of magnitude more efficient than previous approaches. Next, we present an efficient solution for the challenging problem of computing topological changes in detailed dynamic surface meshes. We then introduce a new physics-inspired surface tracking algorithm that is capable of preserving arbitrarily thin features and reproducing realistic fine-scale topological changes like Rayleigh-Plateau instabilities. This physics-inspired surface tracking technique also opens the door for a unique coupling between surficial finite element methods and volumetric finite difference methods, in order to simulate liquid surface tension phenomena more efficiently than any previous method. Due to its dramatic increase in computational resolution and efficiency, this method yielded the first computer simulations of a fully developed crown splash with droplet pinch off.

CHAPTER I

INTRODUCTION

Animations of natural phenomena have become increasingly complex over the past decade. The recent popularity of physics simulation research is mainly due to the constant demand for more visual detail from the special effects industry, the increasing availability of powerful computational resources, and the rapid maturation of algorithms for physics simulation. For example, after Stam [134] introduced the semi-Lagrangian advection scheme to the computer graphics community, fluid animation research abruptly expanded from single-phase smoke simulations [49] into a large array of techniques, equipping a standard Eulerian fluid solver with the ability to produce vastly diverse fluid animations. Eulerian fluid simulations can now simulate large-scale liquids [53], viscous honey [30], viscoelastic jelly [55], bubbles and foam [72], sand [159], and tiny water droplets [151]. Unfortunately, due to the tightly-coupled nature of such numerical simulations, every new physical phenomenon has the potential to impose unacceptably strict memory and time-step restrictions. Such constraints effectively reduce the practical applicability of these animation techniques and prevent the simulation of visually complex scenarios.

1.1 Physics Simulation with Surfaces

This rapid increase in computational complexity is especially apparent when considering physical phenomena with moving surfaces. In order to realistically simulate tiny droplets of liquid, for example, one must overcome the huge memory requirements necessary to resolve such small features on a volumetric fluid grid, as well as the stiff numerical system imposed by large surface tension forces.

Such systems have a unique feature that we can exploit, however. When evaluating the correctness or believability of an animation of such surface-based physical phenomena, we tend to infer the existence of an entire volumetric physical system (a pool of water, for example) by only viewing its surface (the interface between the water and the air). Because we ultimately evaluate the quality of such animations by the appearance of the visible surface, it makes sense to place significantly more importance on the surface of the simulation than the invisible volume.

The aim of this dissertation is to explore the potential gains in computational efficiency and visual fidelity when we de-couple the simulation of the visible surface from the rest of the physics simulation. In particular, if we use a highly-detailed Lagrangian triangle mesh for the surface of the simulation, then we can easily integrate it into simulations based on the Lagrangian finite element method (FEM) or Eulerian finite difference methods. Using this embedded surface mesh, we have seen vast gains in simulation speed and visible detail, as well as the ability to simulate phenomena that were never before possible with other techniques, like a full crown splash with tiny droplets detaching from it (Chapter 8).

1.2 Lagrangian Finite Element Simulation with an Embedded Surface Mesh

In Chapter 3, I will show how to significantly increase the believability, speed, and robustness of a viscoelastic finite element simulation by embedding a high-resolution surface mesh into a low-resolution tetrahedral finite element mesh used for computing the physics. This simulation improves upon previous methods by preserving high resolution

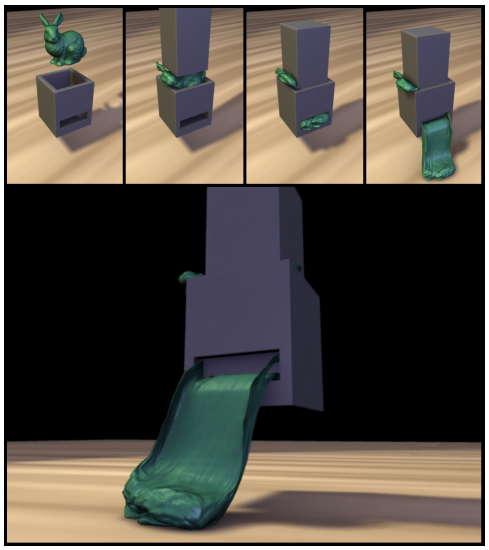


Figure 1: Fast squishing behavior

surface details through low-resolution physical deformations.

1.3 Eulerian Finite Difference Simulation with an Embedded Surface Mesh

In Chapter 4, I will discuss a basic method for integrating an embedded surface mesh into a popular method for Eulerian fluid simulation. In addition to providing an overview of the basic fluid simulation strategy and the methods for maintaining the embedded triangle mesh, I will discuss additional details such as to couple the surface mesh to the fluid simulation, and how to deal with boundary conditions at the free surface.

1.4 Topology Changes in Dynamic Surfaces

After experiencing sufficiently large surface deformations, the surfaces of natural physical systems often exhibit topological changes — they either merge together or split apart. Chapter 5 will provide an introduction to this idea and explain why topological changes are important in nature as well as in animation. In addition, I will provide a short analysis of existing problems when computing topological changes in dynamic surfaces.

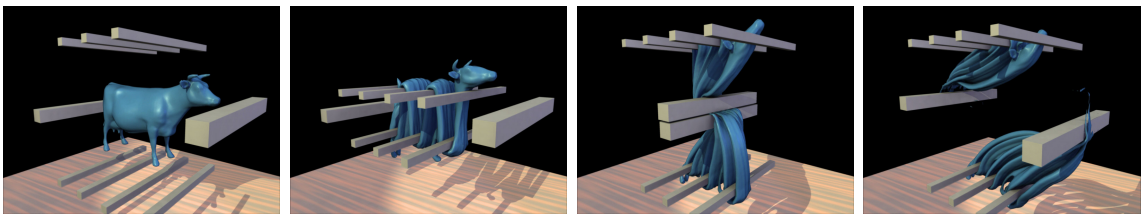


Figure 2: A jiggly bovine that splits and merges

1.4.1 Matching the Topology of an Implicit Surface

In Chapter 6, I will introduce a method that reduces the complicated problem of computing topological changes in an explicit surface mesh to the much easier problem of computing topological changes in a voxelized implicit surface. I will then show how

to integrate this method for dynamic topology changes into elastic, visocelastic, and fluid animations.

1.4.2 Physics-Inspired Topology Changes

In Chapter 7, I will discuss how to compute dynamic topological changes in an Eulerian fluid simulation without introducing serious unphysical artifacts. I will first analyze the stability of each possible fluid structure and use the results of this analysis to inspire a model for dynamic topological changes. The resulting method creates more natural simulations by allowing arbitrarily thin sheets of liquid.

1.5 Layered Surface Tension Simulation

At small scales, surface tension forces significantly influence the behavior of a liquid by producing additional capillary waves, droplets, and bubbles. Such forces tend to act quickly and at high frequencies, so many methods for creating such small-scale water animations exhibit numerical stability problems that are intimately related to these surface tension phenomena. Consequently, either the time step for such simulations must be strictly limited, or the surface tension effects must be heavily damped out by an implicit integration scheme. In Chapter 8, I will outline a strategy for eliminating this dilemma by adding an additional simulation layer to the simulators described in Chapters 6 and 7. With this additional simulation layer, our simulations elegantly reproduce many natural effects like Rayleigh-Plateau instabilities and crown splashes, leading to highly detailed and efficient simulations of surface tension dynamics.

1.6 Contributions

The simulation methods discussed in this dissertation provide several contributions to the computer animation community. These contributions are as follows:

- **Physics-based deformation with Detailed Surfaces** We present a novel combination of an embedded surface mesh with several simulation schemes. We

avoid re-sampling the surface everywhere except where topological changes occur. In those locations, both our local convex hull operation and a subdivision-based method for stitching together surfaces preserve as many Lagrangian surface details as possible. In places where we cannot avoid re-sampling the mesh, we apply an interpolating subdivision scheme to ensure a highly accurate surface. As a result, our method preserves surface features while avoiding common visual artifacts such as popping, smoothing, and volume loss.

- **Resolution of thin features** The algorithms presented in this dissertation can plausibly animate thin sheets and strands of material without lowering the resolution of the simulation. We have not seen such a combination of detailed results and fast simulation times previously in the community.

- **Accurate sub-element mass computation** We introduce a method for computing finite element masses at the resolution of the embedded surface mesh, which is essential for plausible animation of high-resolution surface details.

- **Efficient Tetrahedral Re-meshing** We offer a unique combination of an embedded surface mesh with nonconformal tetrahedral re-meshing, resulting in an order of magnitude speedup over previous approaches, and a related tactic for reducing the time spent transferring simulation data between tetrahedral meshes by exploiting mesh structure.

- **Large range of simulated materials** The technique presented in Chapter 3 efficiently simulates nearly-rigid bricks, dripping slime, and splashing liquid without the need to change numerical solvers or simulation techniques.

- **Mesh topology changes based on an implicit surface** We present a grid-based algorithm for performing topological splits and merges (Chapter 6), which

decreases the memory and computation requirements of dealing with complicated surfaces and permits interesting behaviors like the merging of water droplets.

- **Local convex-hull algorithm for topological operations** We also present a convex-hull based algorithm for computing mesh topology (Chapter 7), which easily preserves thin features and physically-plausible fluid geometries.

- **Constrained Topology** The algorithm in Chapter 7 outputs a high resolution surface mesh whose topology can constrained to match that of a lower resolution fluid grid. This construction allows for the efficient simulation of highly detailed surface animations while preventing many potential artifacts caused by inconsistent mesh and grid topologies.

- **Modular surface tracking:** Our surface tracker does not depend upon any particular simulation technique. In particular, we demonstrate results from both an Eulerian fluid solver and a finite element simulator.

- **Sub-grid surface tension** We introduce a novel method for efficiently simulating detailed surface tension effects that computes wave dynamics on the discretized fluid surface.

- **Stable grid-based surface tension simulation** We provide an algorithm for computing surface tension flows that has a significantly relaxed time step restriction in comparison to previous approaches in graphics.

- **Surface tension approximations** We present two steady-state approximations to subgrid surface tension using mesh-based mean curvature flow (Chapter 3) and volume-preserving mean curvature flow (Chapter 8).

- **Volume preservation** We offer a method for explicitly enforcing volume preservation in a physical simulation, even at scales smaller than the simulation resolution.

- **De-coupled simulation resolutions** We decouple the physics, topological, and surface detail resolutions in our simulator. By modifying the level of detail in each of these behaviors, we can customize our simulations to exhibit complicated phenomena with a limited amount of computation.

Next, in Chapter 2, I will discuss work done by other researchers to address similar problems.

CHAPTER II

PREVIOUS WORK

Throughout this dissertation, we will discuss topics closely rated to computer graphics, computational physics, and computational geometry. The aim of this chapter is to review the relevant previous work done in these areas. We will start with a brief discussion of popular numerical methods used for simulating elastic, plastic, and fluid-like physical phenomena, followed by an overview of prior research regarding the simulation of viscoelastic behaviors and surface tension phenomena. Afterward, we will review previous methods for tracking dynamic surfaces within a physics simulation and extracting a surface mesh from volume data. Because the work in this dissertation utilizes several concepts in geometry and topology, we will conclude this chapter with an overview of related works in computational geometry, specifically in the topics of isosurface extraction, discrete calculus, mesh generation, and computational topology.

2.1 Numerical Methods for Physics-based Animation

In order to animate natural phenomena, many researchers choose to borrow numerical techniques from the applied mathematics community. Primarily, these techniques are used to approximately solve the partial differential equations responsible for the dynamics of a particular physical system. This section will discuss the most popular methods used for simulating physics for computer animation and give a rough idea of which types of physical phenomena are most appropriate for each method.

2.1.1 Eulerian Finite Difference Methods

Eulerian finite difference methods are most often used to simulate fluid-like behaviors. Fluid simulations in computer graphics were first demonstrated by Kass and Miller in [69]. The approach we are using for our liquid simulations in Chapters 6, 7, and 8 was pioneered by Stam [134] and Fedkiw et al. [50, 47]. The combination of operator splitting, semi-Lagrangian advection, and particle level set based surface tracking was afterwards extended to couple with thin shells [57], treat interface discontinuities [60], and handle multiple interacting liquids [90]. A more accurate coupling with solid objects was presented by Batty et al. [9]. Some approaches reduce high computation requirements with model reduction [147] or procedural models for detail [77, 105, 125]. An excellent overview of the algorithms that we use can be found in the book by Bridson [21].

2.1.2 Lagrangian Finite Element methods

To simulate elastically deformable objects, researchers often use finite element methods (FEM). Terzopolous et al. [142, 143, 144], pioneered the idea of using deformable models in computer graphics. O'Brien and Hodgins [111] used a finite element method to animate elasticity and brittle fracture. Later, they added a limited amount of plasticity to the model to produce ductile fracture [110]. Müller et al. [101] introduced a co-rotational formulation for stable FEM animation, and Müller and Gross [99] added plasticity and fracture to the model. Irving et al. [63] made FEM simulations more stable by allowing for invertible elements, and Irving et al. [64] enforced an incompressibility condition.

2.1.3 Meshless Methods

Meshless methods are an interesting alternative to finite elements. Müller et al. [103] applied a meshless method to elastic and plastic simulations, and Pauly et al. [114]

showed that these methods are excellent for computing crack fronts in fracture simulations. Keiser et al. [71] simulated both liquid and solid material with a meshless method. Meshless methods have also been used to simulate melting objects [144] and liquids [100, 159, 2].

2.2 Extraordinary Physical Phenomena

The physics simulation methods described above are most often used to simulate fluids (with Eulerian finite difference methods) and elastic solids (with Lagrangian finite element methods). We are concerned with material behaviors that are not perfectly suited for either of these methods alone. In particular, viscoelasticity bridges the gap between elastic and fluid behavior, while surface tension phenomena is best simulated with a hybrid method combining both Lagrangian and Eulerian qualities.

2.2.1 Viscoelasticity

Chapter 3 of this dissertation addresses the simulation of elasticity, plasticity, and viscosity. Clavet et al. [33] produced elastic and plastic behaviors with a particle simulation, while Goktekin et al. [55] added elasticity to an Eulerian viscous fluid simulation. Losasso et al. [90] extended this model by accounting for the rotation of elastic terms. In our simulations in Chapter 3, we use the method of Bargteil et al. [7], who added large plastic flow to a finite element simulation by recomputing basis functions as they become ill-conditioned.

2.2.2 Surface Tension

Chapter 8 of this document describes a method for the simulation of surface tension flows, though many other researchers have sought to simulate surface tension phenomena in the past as well. The method of Kang et al. [68] is a popular choice for computing the surface tension boundary conditions in Eulerian fluid simulations

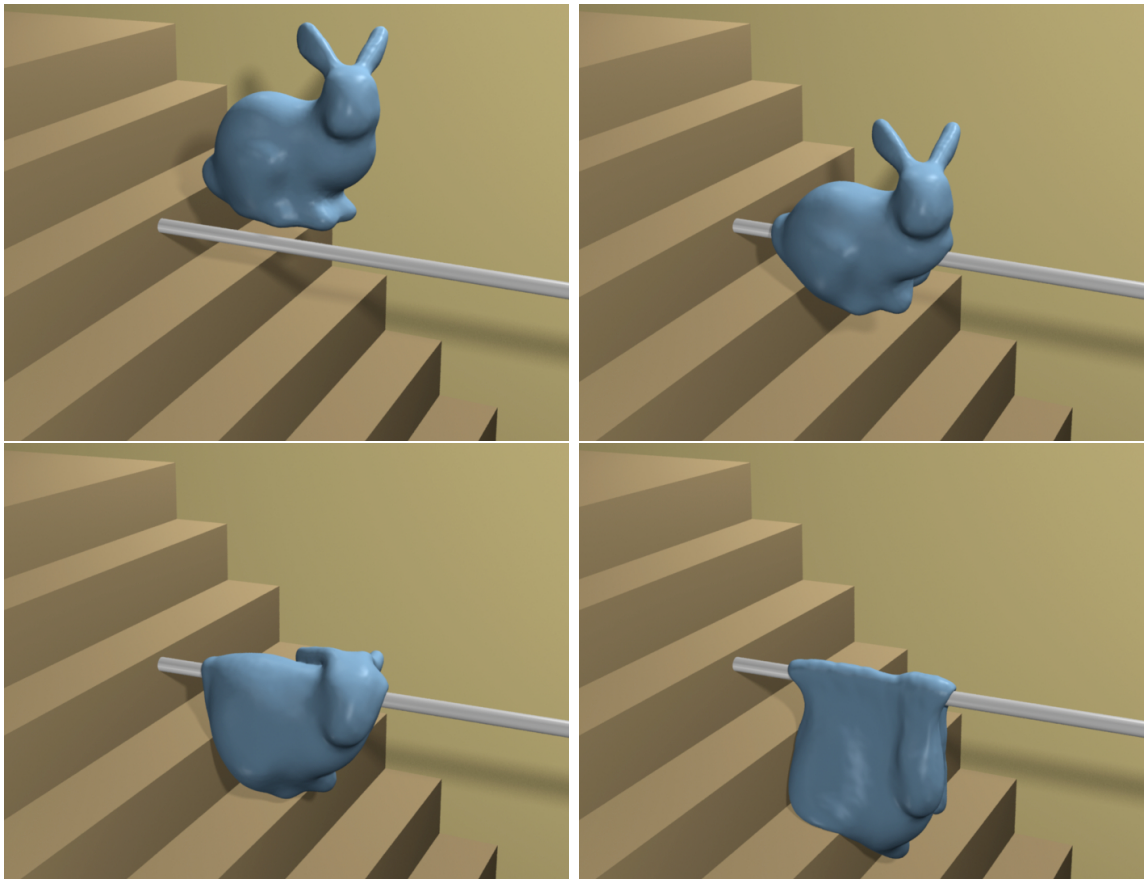


Figure 3: A droopy viscoelastic bunny animated with Lagrangian finite elements [7]

using finite difference methods. Discontinuous boundary conditions across the liquid-air interface have been addressed in [60], while [100] discuss surface tension forces for smoothed particle hydrodynamics (SPH) simulations. Because purely Eulerian surface tension simulations do not behave properly at low grid resolutions, [89] used an octree data structure to simulate surface tension effects. Contact angles of liquids with surface tension are discussed by [151], while [150] use a shallow water solver with implicit surface tension to compute the motion of drops and streams of liquid.

Specialized techniques have been proposed for simulating surface tension phenomena such as drops and bubbles. Bubbles and frothing liquids were simulated in [34], [72] focused on volume control for foam structures, and [76] presented a fast model

for strongly bubbling fluids. [158] introduced a regional level set method with a semi-implicit surface tension scheme for simulating bubbles. A model for both drops and bubbles, motivated by the Weber number of the fluid, was proposed by [93].

One inherent difficulty in direct simulations of surface tension flows is the strict limitation of the simulation timestep. This topic was addressed in [35], who performed multiple surface tension steps per solver iteration. This approach better resolves the surface tension dynamics in time while safely taking larger time steps in the rest of the fluid simulation, though it does not properly couple the surface tension and pressure forces. Recently, Sussman and Ohta [137] proposed a method to relax the time step restrictions from $O(\Delta x^{3/2})$ to $O(\Delta x)$ by performing a volume-preserving mean curvature flow for the computation of the surface tension boundary conditions. This method was also extended to handle contact angles with solid objects [138].

2.3 Surface Tracking

The goal of *surface tracking* is to compute the location of a dynamic surface as it moves through a velocity field. This problem is important within a variety of research areas, from numerical simulations to image processing problems [70]. Currently, the most commonly used technique for tracking such surfaces are level set methods. Level set methods are especially popular in the field of physically based animation, as they can naturally handle topological changes and do not require a surface parametrization. Level set methods track the surface implicitly as the isosurface of a higher dimensional function, typically a signed distance function stored in a Eulerian grid. They were first presented by Osher and Sethian [113], and have since been extended and refined in many ways. Adalsteinsson and Sethian [1] presented a method to restrict the necessary computations to a narrow band, while Sethian [129] developed the related class of fast marching methods. To accurately advect the implicit function, higher-order *WENO* schemes are typically used [87]. However, these more accurate variants of

the level set approach require small time steps to ensure stability. Particle level set methods, developed by Enright et al. [47], combine Lagrangian advection with Eulerian level set schemes to achieve a more accurate surface evolution. Semi-Lagrangian advection of level sets, such as the method by Selle et al. [127], allows arbitrarily large time steps in a simulation. Closely related to level sets are the so-called *volume-of-fluid* methods, that explicitly track a surface by computing mass fluxes [58, 136], but they are not often used in computer graphics applications due to flotsam and jetsam artifacts.

Strain [135] formulated semi-Lagrangian contouring (SLC) for surface tracking, and Bargteil et al. [6] extended the idea for 3D fluid animation. Like our method presented in Chapter 6, SLC computes signed distances exactly from a triangle mesh. However, the accuracy of SLC is limited by the resolution of the implicit surface representation. SLC re-samples an explicit surface by advecting an implicit surface and then reconstructing a triangle mesh, while the related method by Müller [98] advects an *explicit* surface and then reconstructs the mesh from a sampled signed distance function. Müller's method also modifies a set of tables used for surface extraction in order to support thin geometry. Our surface tracker in Chapter 7 uses a similar idea to maintain more complex topology within a grid cell.

2.3.1 Embedded Triangle Meshes

Purely mesh-based surface trackers [121, 54, 66], on the other hand make it more difficult to account for topological changes. McInerney and Terzopoulos [91] use an underlying grid to compute topology changes for mesh-based tracking, but they are restricted to a strictly inward or outward motion. Other approaches have been suggested that focus on detecting and re-meshing self intersections using a set of heuristic rules [19, 81, 24].

One way to present the illusion of a faster, more detailed FEM simulation is to

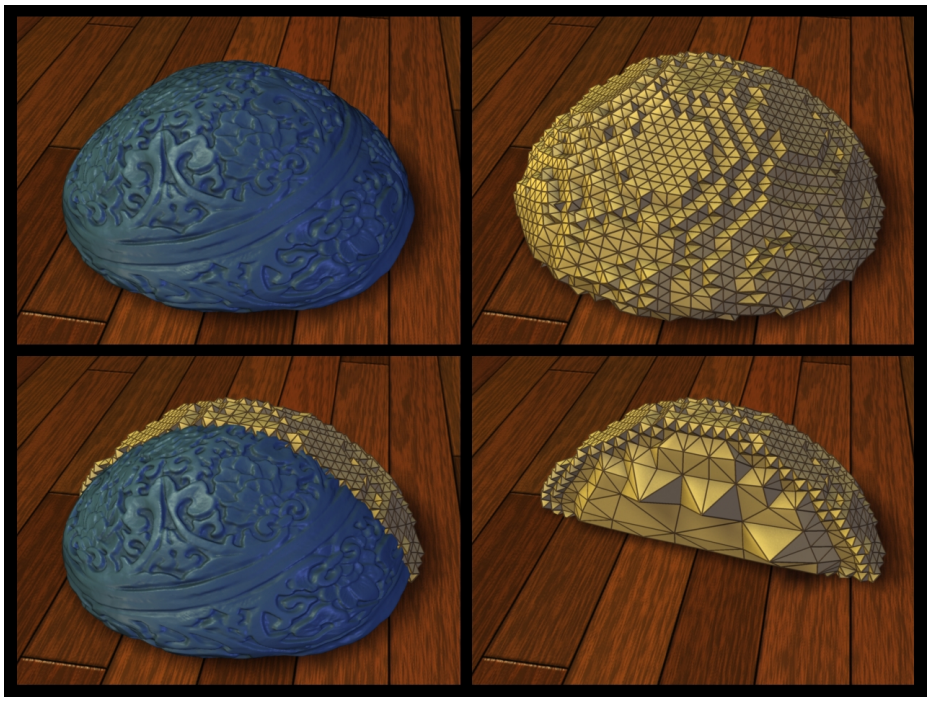


Figure 4: High resolution triangle mesh (blue) embedded into a low resolution tetrahedral mesh (yellow)

embed a detailed surface into a coarse control mesh. Sederberg and Parry [126] introduced this idea of free-form deformation (FFD), and Faloutsos et al. [48] introduced dynamic FFD for animation. Capell et al. [28, 29] used a coarse FEM mesh to control a fine surface mesh, which gives the illusion of a high-resolution elastic simulation. Müller et al. [104] embedded a surface mesh into a finite element simulation to simulate fracture in real time. Molino et al. [95] applied FFD to the crack front of a fracture simulation, skirting around the stability problem that arises with small or poorly-shaped elements. Sifakis et al. [131] improved upon this method by allowing arbitrary cuts within an element. Sifakis et al. [132] developed a method that embedded a high-resolution point-sampled surface in a coarse finite element mesh. They performed collision detection and response using the high-resolution surface by applying forces to the embedded particle directly and updating the finite element mesh using soft bindings.

In addition to embedding a surface into a finite element simulation, Müller et

al. [102] showed elastic behavior can be driven through shape matching. Rivers and James [122] made this idea even more efficient, and Botsch et al. [18] applied this idea to shape modeling. Galoppo et al. [51] used deformation textures to simulate high resolution surface detail with a simplified interior model.

2.4 Implicit Surface Extraction

Chapters 6 and 7 use techniques to convert volumetric data into an explicit triangle mesh surface. Lorensen and Cline [88] constructed a surface from volumetric data using an isosurface extraction technique known as *marching cubes*, which uses a lookup table to generate piecwise linear surface geometry. Because the original marching cubes algorithm produced holes in the surface mesh, several researchers proposed strategies for making it topologically consistent [108, 97]. Another way to ensure that marching cubes templates have a consistent topology is to generate the lookup tables using a convex hull. Bhaniramka et al. [12, 13] showed that convex hulls can be used to reconstruct surfaces for volume data in any dimension, which is one of the reasons we found convex hulls so appealing for our algorithm in Chapter 7. Kobbelt et al. [79] improved the marching cubes method by allowing it to reconstruct sharp features, Ju et al. [67] used dual contouring to construct a surface from hermite data, and Schaefer and Warren [124] proposed a method for contouring the dual of the marching cubes data in order to reconstruct thin structures.

2.5 Calculus on a Triangle Mesh

All of the techniques presented in this dissertation simulate partial differential equations while representing the surface with a triangle mesh. In several instances, we need to communicate geometric shape information from the surface to the simulation or solve differential equations on the surface itself. The fields of discrete differential geometry [92, 40] and mesh processing [17] are especially appropriate in these situations.

In particular, we make extensive use of the discrete Laplace operator. Taubin used an umbrella operator to smooth a triangle mesh [139], while Desbrun et al. [39] used a scale-dependent operator and an operator with cotangent weights [117]. Wardetzky et al. [152] came to the interesting theoretical conclusion that no individual discrete Laplace operator can have all of the desirable properties of the continuous Laplacian.

2.6 Tetrahedral Mesh Generation

The work in Chapter 3 utilizes existing techniques for creating tetrahedral meshes for finite element simulations. The meshing technique by Alliez et al. [3] employs iterative optimization to conform to a surface mesh. Molino et al. [96] used a BCC lattice and a physics-based optimization strategy to enforce mesh conformity. Labelle and Shewchuk [80] also used a BCC lattice and placed guaranteed bounds on the quality of the resulting finite elements. Chentanez et al. [31] use this technique to animate liquid. Most recently, Tournois et al. [146] interleaved Delaunay refinement and optimization to generate high quality tetrahedral meshes.

2.6.1 Topological Changes in Computer Graphics

In Chapter 6, we describe a mesh-based surface tracking technique that computes topological changes by locally replacing the invalid parts of the mesh with one generated by an isosurface creation method, such as marching cubes [88]. McInerney and Terzopolous [91] developed a grid-based method for imposing topological changes upon a mesh that is constrained to surface-offsetting motions. Lachaud et al. [82] merge and split triangle meshes by explicitly sewing and cutting the mesh when nodes come close together, while Zaharescu et al. [157] sew together the exact intersections of overlapping meshes. Pons and Boissonnat [118] use a restricted Delaunay triangulation to sew meshes together. The work most similar in spirit to ours is that of Jian Du et. al [42]; their approach locally replaces invalid geometry with an isosurface. A related technique by Bischoff and Kobbelt [14] converts a CAD model to

a polygon mesh by sewing together closed polygonal patches.

This problem of computing topological changes between triangle meshes is also similar to the problem of computing the solid boolean union between the solid two triangle meshes. Bernstein and Fussel [11] propose a method for computing fast linear booleans, and Campen et al. [27] improve upon this method with an octree acceleration structure.

The control of topological changes is also important in image segmentation, where the topology is known beforehand and should not change. This can be done using concepts from *digital topology* [123]. They have more recently been used to control topology changes of level sets [15]. The algorithm presented in Chapter 6 is based on the complex cell test of Varadhan et al. [149], which tests whether a triangle mesh is topologically equivalent to an implicit surface, except we modify it to retain fine details on the tracked surface.

The next chapter of this dissertation will discuss a method for efficiently animating highly deformable physical behavior. It relates to many of the topics reviewed in this chapter, such as finite element methods, viscoelasticity, surface tension, embedded surface meshes, and mesh generation.

CHAPTER III

LAGRANGIAN FINITE ELEMENT SIMULATION WITH AN EMBEDDED SURFACE MESH

Within the field of computer animation, several researchers have proposed the idea of embedding a high resolution surface into a lower resolution deforming volumetric mesh in order to make a shape animate through time. Such embedded techniques for animating deformable bodies are popular because they give the illusion of highly detailed physics with relatively simple computations. Unfortunately, the nature of these embedded deformations limits the scope of realistic material behaviors to those that will not significantly alter the original embedding. For example, these techniques cannot be used to animate liquid behavior, because the necessary deformations would be too extreme and permanent to capture with such a simple low-resolution volumetric mesh with fixed connectivity. This chapter presents a technique [156] for removing this obstacle and greatly enhancing the range of materials that can be simulated with embedded deformations.

The heart of our technique is a finite element method (FEM) for simulating elasticity and plasticity. We represent the surface of our object with a triangle mesh and embed it into the mesh of tetrahedral elements. Though embedded methods have existed in the literature for years (see Chapter 2), no research has combined such a technique with frequent recomputation of the coarse control mesh. By re-meshing our underlying FEM mesh whenever the simulation quality degrades, we remove several of the barriers preventing embedded mesh techniques from simulating highly plastic behavior.

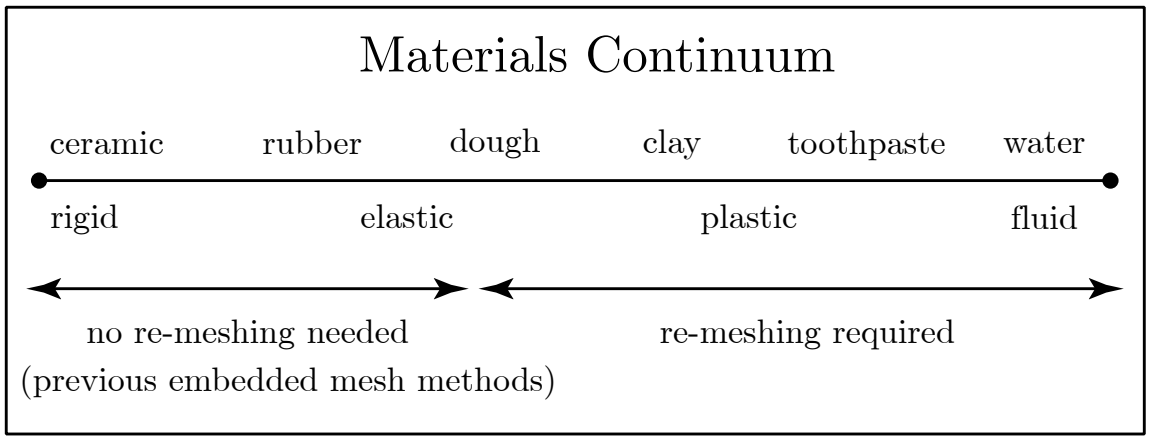


Figure 5: Previous FEM methods with an embedded surface mesh only simulate a limited range of materials because ill-conditioned basis functions result from large plastic flow. Our method recomputes basis functions by re-meshing throughout the simulation, allowing us to simulate a much greater range of materials.

In Figure 5, we display a simplified map of the types of materials useful to computer animators. Materials with limited plasticity, like ceramic and rubber, can be modeled with existing embedded FEM techniques, but extremely plastic behaviors like those of water and toothpaste are not possible due to the fixed topology of the control mesh. In this chapter, I will show how to combine an FEM-driven embedded surface mesh with fast and robust re-meshing, allowing for efficient and stable simulation of behaviors previously unattainable by embedded mesh techniques. In addition, the embedded nature of our method preserves significantly more surface details than existing methods for animating viscoelastic flow, and it remains stable when simulating thin features that have eluded previous techniques. Figure 6 shows an example.

We calculate the continuum mechanics forces on an underlying FEM mesh, taking advantage of our knowledge of the detailed surface whenever possible. We then calculate other forces on the surface mesh, including those from collision resolution and surface tension. Finally, we couple these forces together to animate the surface through time. With an embedded surface, we are free to use a non-conforming FEM

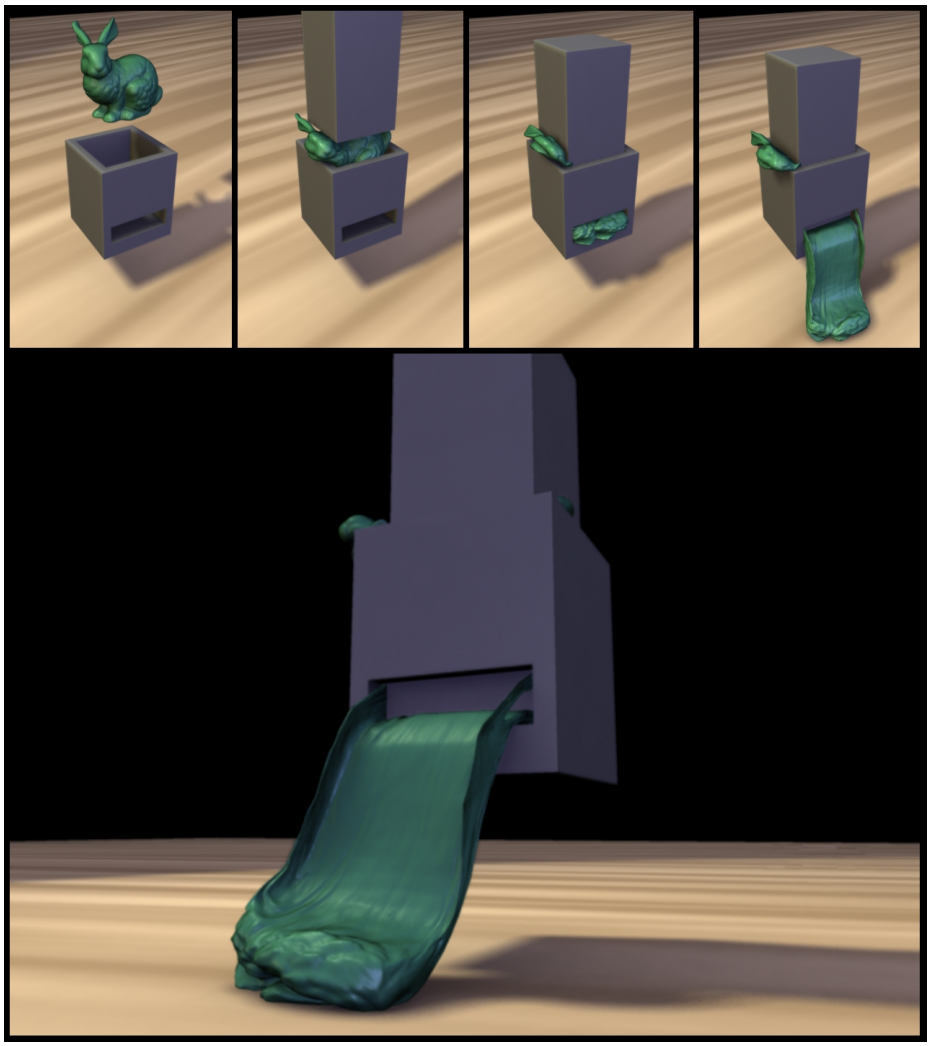


Figure 6: A stiff plastic Stanford Bunny is forced through a small pressing machine that squishes it into several thin sheets. This result was generated in 30 minutes using a re-meshing finite element method with an embedded surface mesh.

mesh. We use the Delaunay tetrahedralization of an un-warped body-centered cubic (BCC) lattice as our FEM mesh, and we exploit its regularity for much faster mesh creation and point location queries. We sidestep inaccuracies from this low resolution mesh by carefully computing nodal masses whenever necessary.

3.1 Physical Model

We wish to simulate material behaviors that exhibit *elasticity* and *plasticity* in our animations. These terms describe how a material reacts after it has been deformed

and all stress has been removed. A purely *elastic* material will return back to its original shape after stress has been removed; all deformations to an elastic material are perfectly reversible. A rubber ball is a good example of an elastic material. A *plastic* material, on the other hand, will have permanent deformation after any stresses have been removed. Metal is an example of material that exhibits some plasticity – metal will permanently bend after being hit with a hot hammer in a forge. Liquid is a material that can be considered completely plastic. There are no elastic forces attempting to return it to its original shape after a deformation.

When we discuss *viscoelastic* behavior, we are talking about materials that exhibit both viscous (internally damped) and elastic behavior, like honey or jelly. *Viscoplastic* behavior refers to materials that exhibit rate-dependent plasticity. For example, a viscoplastic material with a low flow rate may elastically deform due to large sudden deformations and flow plastically under small, slow deformations.

To produce our animations of elastic, plastic, viscoelastic, and viscoplastic phenomena, we use the elasticity model of Irving et al. [63] and the plasticity model of Bargteil et al. [7]. This plasticity model incorporates creep and work hardening:

$$\hat{\mathbf{F}}_p = (\hat{\mathbf{F}}^*)^\gamma, \tag{1}$$

$$\gamma(\mathbf{P}, P_Y, \nu, \alpha, K) = \min\left(\frac{\nu(||\mathbf{P}|| - P_Y - K\alpha)}{||\mathbf{P}||}, 1\right) \tag{2}$$

where $\hat{\mathbf{F}}_p$ is the diagonalized plastic deformation tensor, $\hat{\mathbf{F}}^*$ is the volume-conserving portion of the diagonalized deformation tensor, $\mathbf{P}$ is the stress tensor, P_Y is the plastic yield point, ν is the flow rate, K is a hardening parameter, and α is a scalar representing the accumulated plastic stress. γ is a function that determines the amount of plastic flow each element will experience: an element with γ equal to zero will have purely elastic behavior, while an element with γ equal to one will behave purely plastically. See Bargteil et al. [7] and Irving et al. [63] for more details.

Following Bargteil et al. [7], we allow for arbitrarily large plastic flow by re-meshing

whenever the finite element basis functions become ill-conditioned. After creating a new FEM mesh, we copy all old simulation variables to the new, well-conditioned mesh through interpolation and averaging. Unlike previous work we employ an explicit surface mesh, avoiding problems such as volume loss and limited surface detail. We also use a non-conforming tetrahedral FEM mesh based on a BCC lattice [96, 80] (that is, the boundary of the finite element mesh does not precisely align with the material interface). To advance our simulations forward through time, we use a variable time step Newmark integrator, as in Bridson et al. [20].

3.2 Embedded Surface Mesh

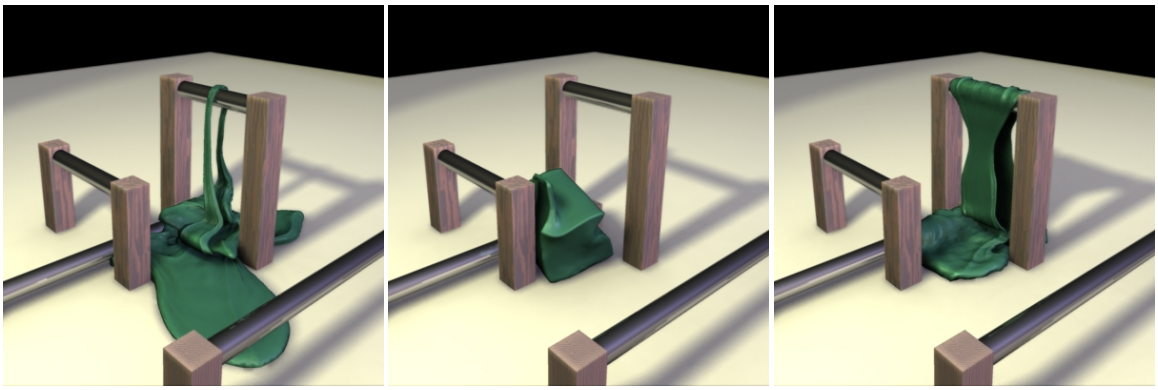
Our method significantly differs from previous work in the treatment of the surface details of an object. Instead of using the boundary of the FEM mesh as the exterior surface of our simulated material, we provide a separate explicit mesh for the surface. We use this surface mesh for rendering as well as for collision dynamics, but all of the elasticity and plasticity computations come from the finite element simulation.

For each vertex in the surface mesh, we find the tetrahedral element that completely encloses it, and we assign the vertex to that tetrahedron. From this point on, we can express the surface vertex position $\mathbf{x}$ as a linear combination of the position of tetrahedral vertices $\mathbf{x}_i$ using barycentric coordinates b_i:

$$\mathbf{x} = b_1\mathbf{x}_1 + b_2\mathbf{x}_2 + b_3\mathbf{x}_3 + b_4\mathbf{x}_4. \tag{3}$$

This way, when the finite element calculations distort the parent tetrahedron, the surface mesh will distort as well. We can also express the velocity and forces on this surface vertex as a linear combination of the velocities and forces of the parent element's vertices using these same barycentric coordinates.

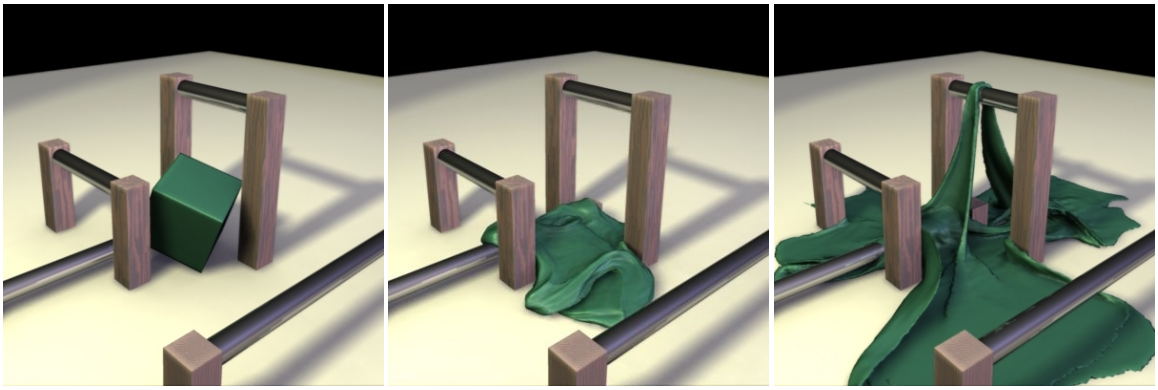
For collisions with objects in the environment, we use a projected-vertex approach [63]. However, instead of computing collisions with FEM nodes, we compute collisions with respect to the embedded surface vertices. If a vertex is involved in



$\mu = 2 \times 10^6$
$\beta = 1 \times 10^3$
$P_Y = 0$
$\nu = 1 \times 10^5$

$\mu = 2 \times 10^6$
$\beta = 3 \times 10^3$
$P_Y = 3 \times 10^4$
$\nu = 1 \times 10^{30}$

$\mu = 2 \times 10^5$
$\beta = 1 \times 10^2$
$P_Y = 3 \times 10^3$
$\nu = 2 \times 10^3$



$\mu = 1 \times 10^7$
$\beta = 1 \times 10^4$
$P_Y = 1 \times 10^{10}$
$\nu = 0$

$\mu = 2 \times 10^6$
$\beta = 1 \times 10^2$
$P_Y = 1 \times 10^1$
$\nu = 1 \times 10^3$

$\mu = 2 \times 10^6$
$\beta = 1 \times 10^2$
$P_Y = 0$
$\nu = 1 \times 10^5$

Figure 7: Material parameters for a dropped cube, demonstrating a range of behaviors ranging from rigid to fluid-like. Each image is taken from a separate animation with different material parameters. μ is stiffness, β is viscosity, P_Y is yield stress, and ν is flow rate.

a collision, we project the vertex onto the obstacle's surface by moving the entire element such that the collision is resolved. To do this, we define distribution weights for each surface vertex:

$$w_i = \frac{b_i}{b_1^2 + b_2^2 + b_3^2 + b_4^2}, \quad i = 1, 2, 3, 4 \tag{4}$$

where w_i is the distribution weight corresponding to node i of the parent tetrahedron, and b_i is its corresponding barycentric coordinate. Note that these weights are the tetrahedral analogy of the weights devised by Bridson et al. [22] for triangles and line segments. Using these distribution weights, we distribute the position change, $\Delta\mathbf{x}$, to each of the nodes of the tetrahedron via the equation $\mathbf{x}_i = \mathbf{x}_i + w_i\Delta\mathbf{x}$. Similarly, in the presence of friction we add velocity impulses, $\Delta\mathbf{v}$, with $\mathbf{v}_i = \mathbf{v}_i + w_i\Delta\mathbf{v}$. To avoid competing collisions within an element, we only apply the collision resolution to the deepest penetrating vertex in each element. For self collisions and collisions with other deformable bodies, we use repulsions and friction from the model of Bridson et al. [22]. We apply impulses to each surface vertex using the distribution weights as described above.

In simulations with large plastic flow, sometimes the surface triangles stretch to be so large that they become visually distracting, or shrink to the point that they are unnoticeably small. To maintain a quality surface mesh, we subdivide long edges at the midpoint and bisect the two incident triangles. We also perform a simple edge collapse operation if any edge in the mesh is too short. If volume preservation is of utmost importance, one might decide to implement a more sophisticated scheme like that of Lindstrom and Turk [86].

Lastly, our embedded scheme allows several additional optimizations: because each surface vertex is guaranteed to lie within its parent element, the tetrahedral FEM mesh doubles as a collection of bounding volumes that can greatly speed up collision detection. As the position of a surface vertex is given by the barycentric coordinates within a tetrahedron, there is no need to update the position of a surface

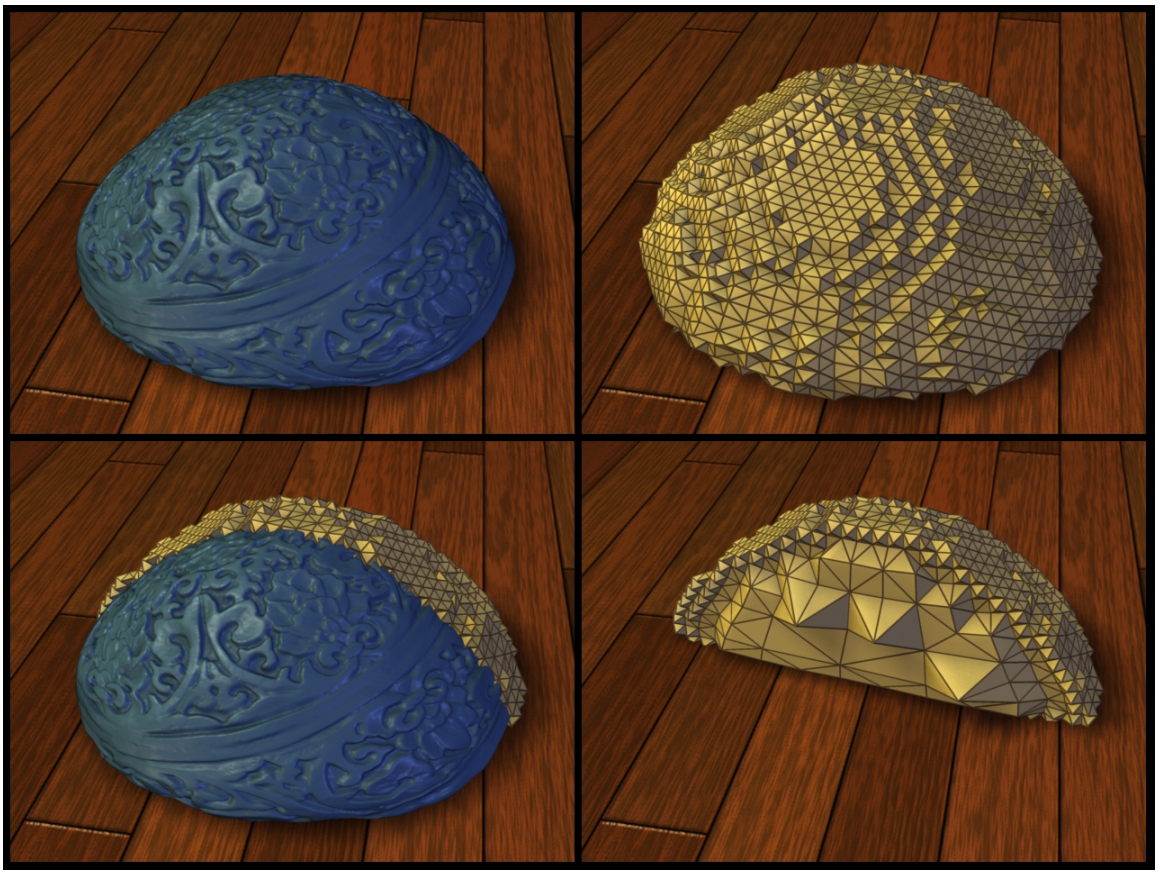


Figure 8: A high resolution surface mesh (blue) embedded into a low resolution adaptive BCC lattice (gold). We show the cross section of the FEM mesh in the bottom row.

vertex unless it is participating in a collision or we are rendering the surface. If there is no collision with the bounding tetrahedron, there is no need to update any of the surface vertices within it. This idea is especially useful when there are hundreds of surface vertices per element, as in Figure 8.

3.3 Re-meshing

We use a tetrahedral mesh constructed from a BCC lattice [96, 80] for our finite element computations. To create the tetrahedralization, we first voxelize our surface mesh onto two offset regular cubic grids. Any voxelization algorithm will suffice; we chose to scan convert the surface mesh by sorting triangles by their x and y bounding boxes, casting rays parallel to the z-axis, and counting triangle intersections up to each grid point. This voxelization method is analogous to polygon rasterization, as noted by Nooruddin and Turk [109]. During voxelization, we classify each lattice node as internal or external with respect to the surface mesh. Finally, we create any BCC tetrahedra that touches an internal node or overlaps any geometry in the surface mesh. For added efficiency, we use graded BCC tetrahedra on an octree as in Labelle and Shewchuk [80].

We embed our surface mesh into the volumetric tetrahedral mesh, so we do not spend any computational effort modifying our BCC mesh to conform to the surface. Consequently, we do not need to perform any iso-surface evaluations. We never move any BCC nodes and we only briefly deal with the surface, yielding a highly efficient meshing algorithm. As Labelle and Shewchuk [80] state, although their technique is already quite fast, the mesh generation step of their algorithm took 1% of the total time, while the rest was spent on isosurface evaluation. Based on their timings, using a nonconforming mesh (as we do) is roughly one hundred times faster than the state of the art in conformal meshing.

After re-meshing, we transfer FEM simulation data from the old tetrahedral mesh

to the new one. In our simulations, this data consists of a deformation gradient tensor $\mathbf{F}$ and a scalar accumulated stress α stored at each element, and a velocity vector $\mathbf{v}$ stored at each node. To assign data to each element in the new mesh, we compute the average of the data from each overlapping tetrahedron in the old mesh, weighted by the amount of overlap in volume. To transfer node-based data, we first find which tetrahedron in the old mesh encapsulates the node, then use barycentric interpolation to find the new data. Boundary nodes present a slight complication—we do not use a conforming tetrahedral mesh, so some new simulation nodes will lie outside of the old simulation mesh. Barycentric interpolation is not possible here because there is no bounding tetrahedron. Instead, we find the closest element to the node and use barycentric extrapolation. Extrapolation is valid in this case because it preserves the velocity of the embedded surface vertices. We do not do anything special for new elements that only partially overlap the old tetrahedral mesh; a weighted average of overlapping tetrahedra is sufficient for these edge cases.

For the efficient transfer of data between arbitrary meshes, we could use a kd-tree for point location and a hierarchical system of bounding boxes for tetrahedral-overlap queries. Fortunately, we do not need to resort to any of these data structures for efficient transfer of variables after a re-mesh because we can take advantage of the BCC lattice structure of our new undeformed mesh. We perform point location in logarithmic time using a BCC octree (constant time for a uniform lattice), and we save memory by never allocating any auxiliary data structures. We transfer per-node variables by iterating through each tetrahedron in the old deformed mesh and performing a fast lookup into the undeformed BCC lattice of the new mesh to find out which nodes in the new mesh overlap the tetrahedron's bounding box. For each node from the new mesh that is inside of this tetrahedron, we transfer data using barycentric interpolation. Per-element variable transfer is just as efficient: for each tetrahedron in the old mesh, we can quickly find all new tetrahedra that overlap its

bounding box by taking advantage of the undeformed lattice structure of the new mesh. We compute the volume of overlap between these new tetrahedra and the old one, and use that volume for the weighted average in the per-element data transfer.

This immense savings in time spent re-meshing might be viewed as unimportant because we only re-mesh sporadically throughout the simulation. After all, Bargteil et al. [7] reported that they only spent 13% of the simulation time re-meshing. However, re-meshing is nearly free in our case, so we can afford to re-mesh quite frequently, allowing us to simulate phenomena with a much higher flow rate ν in the same amount of time.

3.4 Exact Mass Computation

We use a lumped mass formulation in our finite element simulation, as described in O'Brien and Hodgins [111]. This entails computing the mass contribution of each element, then distributing that mass to the nodes of the FEM mesh by assigning one quarter of the element's mass to each of its nodes. One problem with blindly applying this strategy to our simulation with an embedded mesh is that the mass per element no longer accurately reflects the actual amount of mass we may wish to represent. This is especially problematic with coarse FEM meshes or thin strands of surface material.

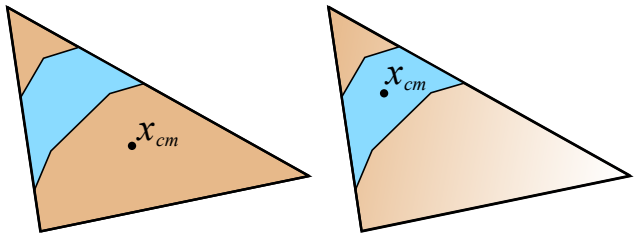


Figure 9: Sub-element mass computation: The blue strip of material is clipped to the element, and then masses are distributed to nodes at the element corners, resulting in a linear density function represented in beige. Naive lumped mass calculations give a uniform density across the element and an incorrect center of mass x_{cm} (left). Our method computes the correct center of mass and density distribution (right).

Figure 10: A viscoelastic armadillo drips through a straining device, resulting in several thin strands of slime. Because we compute nodal masses at sub-element resolution, our method maintains physical plausibility in the presence of these small features. This result was simulated at 30 seconds per frame

For example, in the left side of Figure 9, a thin strip of material only covers part of the element. By using a naive lumped mass formulation and ignoring the surface mesh,

the center of mass actually exists outside of the material, and the total mass in the element grossly overestimates the amount of mass occupied by the surface mesh. This leads to sudden unexpected movements if the center of mass varies significantly after a tetrahedral re-mesh, and the error in total mass unphysically injects momentum into the system during significant plastic flows. A slightly inaccurate mass and inertial moment may not be visually disturbing by themselves, but they may cause noticeable artifacts if we continually recompute inconsistent physical properties on different FEM meshes through time.

Our goal is to calculate mass distributions per element that accurately reflect the properties of the surface mesh. If we ensure that they are correct on a per-element level, then the physical properties of the overall object will be equal before and after a re-mesh event. Our strategy for computing correct nodal masses is to first compute the exact mass and center of mass occupied by the material in each element, then distribute the mass from the exact center of mass of the material within that element. This will ensure that the mass and moment of the element match the mass and moment of the material within it. Interestingly, Batty et al. [9] used a similar technique in a much different context to achieve sub-grid accurate coupling in an Eulerian fluid simulation.

We compute the mass within an element by clipping the surface mesh to the tetrahedral element, then computing the mass m and center of mass x_{cm} of the resulting closed polyhedron. Our implementation uses the method by Lien et al. [83] to compute the volume V of the clipped surface mesh. We then compensate for the elastic deformation by dividing by the determinant of the deformation gradient $\mathbf{F}$.

$$m = \frac{\rho V}{|\mathbf{F}|} \tag{5}$$

where ρ is the material density. After we have obtained m and x_{cm}, we compute barycentric coordinates $b_{1,2,3,4}$ and distribution weights $w_{1,2,3,4}$ for x_{cm}, and distribute m to each node of the tetrahedron via $m_i = m_i + w_i m$.

Unfortunately, just as poorly-shaped tetrahedral elements can introduce stability problems, so can small nodal masses. The exact nodal mass computation is ideal for accurate simulations, but proves impractical in the presence of thin features, where masses per node may be near zero. We implemented a minimum node mass $m_{\min}$ as a tunable simulation parameter. Each time we re-mesh, we compute the exact mass for every FEM node and then clamp any unacceptably small mass to $m_{\min}$. This allows the user to strike a balance between accuracy and simulation speed. We have found that overestimations of mass were not noticeable for largely elastic simulations, but large plastic flows leading to thin sheets and strands required more accurate masses. In our most fluid-like simulations, we set $m_{\min}$ to be 5% of the mass that would be normally given to a node surrounded by fully-massive elements.

This method of accurate mass computation is reasonably effective, despite its stability problems. After this work was published in 2008 [156], Nesme et al. [106] eliminated this stability problem by additionally reducing each element's elastic stiffness in the presence of small nodal masses.

3.5 Surface Tension Approximation

We occasionally wish to compute surface tension forces as well as elastic and plastic forces. We leverage the work by Brochu [24] to compute momentum-conserving surface tension forces on the high resolution surface. Brochu used triangles as the simulation primitives, while we use vertices, so we distribute per-triangle forces by dividing the force equally among the three vertices of each triangle. The force on a triangle from a neighboring face is a vector in the plane of the neighbor and normal to the shared edge, proportional to the edge length and the surface tension coefficient. So the total surface tension force on each triangle is:

$$\mathbf{f}_T = \sum_{i=1}^{3} \sigma(\mathbf{n}_i \times \mathbf{e}_i) \tag{6}$$

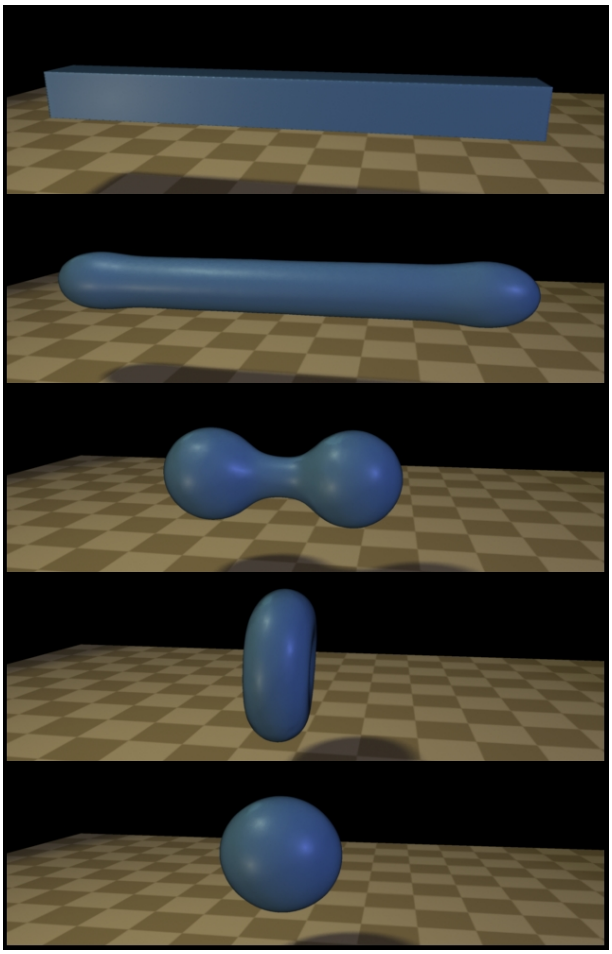


Figure 11: Surface tension smoothly changes the shape of a rectangular block.

where σ is the surface tension coefficient, $\mathbf{n}_i$ is the normal of the neighboring triangle i and $\mathbf{e}_i$ is a vector representing the shared edge between this triangle and this neighbor, with the edge vectors pointing clockwise around the triangle. The surface tension force on each vertex is one third of the total force from all surrounding triangles:

$$\mathbf{f}_{\text{tension}} = \frac{1}{3} \sum_{j=1}^{n} \mathbf{f}_{T,j} \tag{7}$$

where j iterates through each of the vertex's incident triangles. Once we have a per-vertex surface tension force $\mathbf{f}_{\text{tension}}$, we distribute these forces to the FEM nodes using distribution weights ($\mathbf{f}_i = \mathbf{f}_i + w_i \mathbf{f}_{\text{tension}}$). This suffices to produce a plausible surface tension simulation on the FEM mesh.

These coarse surface tension forces do not change sharp features within an element.

```
Calculate forces (elastic/plastic,
                  gravity,
                  surface tension)
Integrate FEM nodes
If SurfaceTension
  Smooth the surface
  Recompute barycentric coordinates
Update surface vertex positions
Collision detection and resolution
Update surface vertex positions
If RemeshNeeded
  Subdivide large surface triangles
  Collapse short surface edges
  Create new FEM mesh
  Recompute barycentric coordinates
  Transfer variables to new mesh
  Re-calculate mass in FEM nodes
```

Figure 12: Pseudo-code for one timestep of Lagrangian FEM simulation

To smooth out local features, we use explicit Laplacian smoothing with a scale-dependent umbrella operator [39]. At each time step, we apply this smoothing to each vertex in order to get a vector $\mathbf{x}_{\text{smooth}} - \mathbf{x}_{\text{old}}$ from its original position $\mathbf{x}_{\text{old}}$ to its smoothed position $\mathbf{x}_{\text{smooth}}$. We then scale it back by a multiple of the surface tension coefficient σ and the time step size Δt:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} + \sigma \Delta t (\mathbf{x}_{\text{smooth}} - \mathbf{x}_{\text{old}}) \tag{8}$$

We used this technique to generate the animation in Figure 11.

3.6 Algorithm Overview

Our algorithm is summed up by the pseudo-code in Figure 12. The timestep begins by calculating all of the forces on each FEM node due to elasticity, plasticity, gravity, and surface tension. Then the velocities and positions of the nodes are updated via time integration. If we wish to simulate surface tension, we then apply geometric smoothing and recompute barycentric coordinates of the surface vertices. Then we

handle collisions and update surface vertices to account for any changes in position. At the end of the timestep, we decide whether we should re-mesh. A re-mesh event can be triggered by ill-conditioned element basis functions or a significant change in the size or shape of an element. We also force a re-mesh if geometric smoothing has caused a surface vertex to move significantly outside of its parent element. The simulations in this paper re-meshed if any element tripled its condition number since the last re-mesh event, or if any barycentric coordinate dropped below -0.2.

Once we have decided to re-mesh, we first operate on the surface mesh by subdividing large triangles or decimating small ones. Then we create a new mesh using the procedure in Section 3.3. Finally, we recompute barycentric coordinates, transfer simulation variables from the old mesh to the new one, and re-calculate FEM masses.

For the sake of clarity, we did not incorporate any optimizations into Figure 12 (such as delaying surface vertex updates as long as possible), and we did not discuss special collision treatment during time integration. For example, our algorithm uses the integrator of Irving et al. [63], which interweaves collision handling and velocity integration.

3.7 Input Parameters

Our computational model requires several parameters to be set by the user. The elasticity model that we use [63] has two stiffness coefficients and (one dealing with the overall stiffness of an object, and one dealing with the volume-preserving stiffness) and two damping coefficients (one dealing with overall visocosity in an object, and one dealing with only rotation-preserving viscosity). In addition, the surface tension model requires a surface tension strength and a geometric smoothing strength. The physical model presented in Section 3.1 names several plasticity parameters that must be chosen properly in order to simulate a specific material behavior. Figure 7 shows many of these physical parameters and may give a better intuition for how to tune

them.

We also need to set a density for the material and a minimum mass per simulation node. Larger values for density and minimum mass tend to stabilize the numerical system.

Our collision detection and resolution is based on the algorithm of Bridson et al. [22]. This algorithm requires setting a repulsion distance, friction coefficients, and a roundoff tolerance, in addition to implementing a bounding volume hierarchy.

We have two meshes in our simulation: a tetrahedral mesh for the finite element calculations, and a triangle mesh for the visible surface. The finite element mesh generation procedure (based on [80]) primarily requires a minimum edge length and an octree depth in order to work properly. We chose this minimum edge length based on how many tetrahedra we wanted in our simulation; more tetrahedra lead to more detailed simulations and slower simulation runs.

We also had to choose parameters based on how to re-sample our surface mesh. We chose a minimum edge length (shorter edges were collapsed at the beginning of each time step) and a maximum edge length (longer edges were subdivided at the start of each time step). These parameters are closely linked to the visible feature sizes on the surface; more detailed surfaces require a smaller minimum edge length, although more details require more memory. The shape of the triangles in this simulation only affects stability when surface tension (geometric smoothing) is enabled.

One thing to note is that the FEM shape functions within each tetrahedron are linear, so a very finely detailed surface mesh will still be restricted to piecewise linear motions based on the size of tetrahedra in the simulation mesh. For purely elastic simulations, a highly refined surface mesh embedded in a low resolution tetrahedral mesh can look distractingly jagged when undergoing large deformations. This problem was unnoticeable for plastic animations which routinely recomputed the tetrahedral mesh.

Finally, the size of a simulation time step is another parameter in our simulation, and we use a variable time step scheme [20] in order to choose the largest stable time step size.

3.8 Results

Figure 8 shows a coarse finite element mesh of 20k tetrahedra used in conjunction with an extremely fine surface mesh of 360k triangles, illustrating how a complex surface works with a nonconformal FEM mesh. In Figure 6, we smashed a 70k triangle bunny model into a ream of thin material. The total simulation time was 30 minutes. We chose this example to show how our method can efficiently simulate conditions that would push other techniques to their limits. The armadillo in Figure 10 started with 8k elements and 71k surface triangles, finished with 11k elements and 113k triangles, and simulated at 30 seconds per frame. Notice how our method preserves the detailed ridges on the armadillo's face, legs, and back, and notice the thin strands of goo as pieces of his legs slowly drip off. The thinnest strand in the simulation is more than one hundred times smaller than the FEM resolution. Figure 13 shows a simulation of two different colored hands colliding with each other. The distinct meshes stick together but do not merge, illustrating how this method can be used to animate immiscible fluids. Figure 7 shows the range of material parameters capable with this technique. By varying the viscosity, plastic yield, flow rate, and elastic stiffness, we were able to generate rigid bricks, thick viscous ooze, jiggly gelatin, gooey jelly, and splashing fluid. On average, each of the examples in Figure 7 took an hour and a half to simulate, while the most extreme fluid example (Figure 7, far right) simulated for about 10 hours. It began with 7.5k elements and 12k triangles, and ended with 60k elements and over 600k triangles. We used NVIDIA's Gelato to render the animations in this chapter. Ambient occlusion was utilized in all animations except for Figures 8 and 10, which caused some shadow popping artifacts in the video.

Though we are proud of the quality and speed of our simulations, our method could greatly benefit from a GPU implementation and a fully implicit integrator. These additions could speed up some moderately complex examples to real time. We can already run some coarse examples at interactive rates on our unoptimized research code. Alternatively, we believe that a professionally-implemented version of this method could generate detailed production-quality results in tens of minutes.

3.9 Discussion and Limitations

We have presented a method for animating a large range of different material parameters with relatively quick simulation times. We designed our method to handle exceptionally thin features without becoming unstable. Our system can even address the thin sheet problem that arises in many fluid simulation techniques. Because the plasticity model forces our tetrahedra to locally preserve volume, volume is approximately conserved even for extremely thin features embedded in the FEM mesh. Figure 7, far right, shows an example of a low-viscosity fluid that has thin sheets everywhere, yet 95% of the volume was retained by the end of the simulation. A grid-based level set method would fail to resolve any of these features, resulting in an aggressive deletion of material. The small amount of volume loss in our animations is largely due to our nonconservative edge-collapse operations. Simulations that do not need to collapse small edges typically retain more than 99% of their volume. If volume conservation were especially important, we could enforce incompressibility in our elasticity simulation [64].

We have found several benefits to using an explicit embedded surface mesh in our simulations. During the simulation of extreme plastic deformation, we must recompute the FEM mesh often in order to maintain stable basis functions. Without our explicit mesh, temporal continuity across re-meshing events would pose a problem:

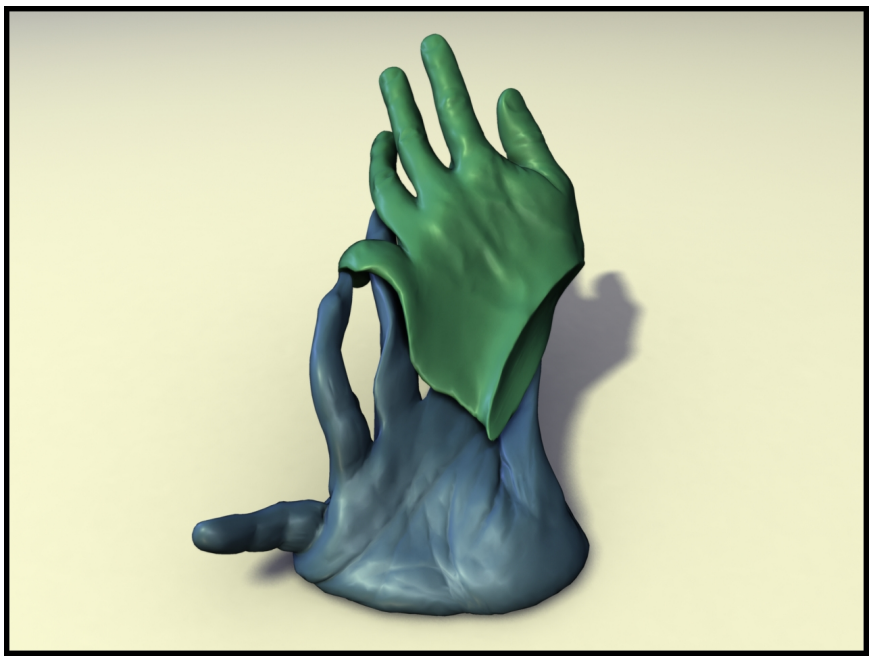


Figure 13: Two gooey hands stick together in mid flight.

visible surface smoothing and popping can occur when changing from one conforming FEM mesh to another, which is annoying at best and catastrophic for collision handling at worst. In contrast, our embedded surface mesh remains C^0 continuous with respect to time.

Our re-meshing algorithm executes quickly, but it also allows large simulation time steps. Because the size of the time step is dictated by the worst element in the mesh [130], and because every element in our mesh starts in the exact same nearly-optimal shape, we are not limited by a single poorly-shaped tetrahedron. Of course this changes if we decrease the mass of the nodes for improved accuracy, but we have presented the idea of a minimum nodal mass as an intuitive control knob for balancing speed and physical realism.

As presented in this chapter, this system lacks the ability to change surface topology. It can simulate immiscible materials (Figure 13), but it cannot simulate several objects merging together without additional modification. We will discuss methods for integrating explicit changes to surface topology later in this dissertation.

To conclude, the method described in this chapter is efficient and capable of simulating a wide variety of materials. With our embedded surface mesh, we can add an arbitrary amount of detail and compute high resolution surface tension forces without increasing FEM resolution. We have also presented an efficient re-meshing algorithm that yields near-perfect tetrahedra, priming the simulation for speed. Our method is also capable of animating surface details like thin sheets and strands that are impossible with other techniques.

CHAPTER IV

EULERIAN FINITE DIFFERENCE SIMULATION WITH AN EMBEDDED SURFACE MESH

In the previous chapter, I presented a method for simulating viscoelastic materials with detailed surfaces by embedding a high-resolution surface mesh inside a Lagrangian finite element simulation. In the Lagrangian reference frame, the computational variables (such as velocity and elastic strain) are stored in computational nodes that deform and translate along with the simulated material. In the absence of a remeshing event, the location of surface vertices can be described with fixed coordinates relative to the computational nodes (the vertices of the tetrahedral elements). Such Lagrangian finite element-based simulation techniques work remarkably well for simulating highly elastic behavior, but they are unnecessarily computationally intensive when simulating highly plastic (fluid) behaviors.

In this chapter, I will explain how to efficiently animate detailed fluid behaviors by embedding an embedded surface mesh within an Eulerian finite difference simulation. As opposed to the Lagrangian framework, the computational variables in an Eulerian reference frame stay fixed in space while the simulated material deforms and translates. Instead of defining surface vertex locations with fixed barycentric coordinates relative to the computational grid, we recompute surface vertex positions every timestep. Such Eulerian grid-based simulation techniques have difficulty simulating hyperelastic behaviors due to excessive re-sampling errors, but they are extremely convenient for simulating plastic fluid behaviors.

4.1 Physical Model

We start by building upon a common technique for simulating fluid dynamics in computer graphics: We aim to simulate the partial differential equations which govern subsonic fluid flow, known as the incompressible Navier-Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{g} \tag{9}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{10}$$

where $\mathbf{u}$ is the velocity of the fluid, t is time, ρ is the density of the fluid, p is the fluid's pressure, ν is the fluid's viscosity, and $\mathbf{g}$ is the acceleration due to gravity. We store these variables on a regular hexahedral grid, with the pressure values stored at the centers of each grid cell and the velocity components stored on the faces of each grid cell. The details of storing these variables are discussed further in [21].

We can think of the Equation 9 as a collection of competing accelerations that push around the fluid. This equation governs the momentum of the fluid, and we will refer to it as the *momentum equation*. Equation 10 is a constraint on the momentum equation that forces the fluid's velocity field to have a divergence of zero ("divergence-free"). When the fluid's velocity has no divergence, it is considered *incompressible*, and there are no sources or sinks of momentum in this system. Additionally, as long as the density of the fluid is constant, this divergence-free constraint implies that the fluid must conserve mass and volume.

To simulate the fluid, we can break up the momentum equation into a series of terms using a technique called *operator-splitting*. This effectively means that each term on the right hand side of Equation 9 can be applied independently, one after the other. To account for the gravity term, we simply add a constant downward acceleration to our velocity field. To account for the viscosity term, we solve the diffusion equation $\frac{\partial \mathbf{u}}{\partial t} = \nu \Delta \mathbf{u}$. The advection term $-(\mathbf{u} \cdot \nabla)\mathbf{u}$, has been studied extensively in the applied mathematics and computer graphics literature, and we can

use any of several techniques ([134, 49, 73, 94]). We used a higher-order variant on semi-Lagrangian advection known as unconditionally stable MacCormack advection [127] for most of our simulations.

Because the pressure field p is unknown at the beginning of each time step, we can use it as a Lagrange multiplier to satisfy the the divergence-free constraint (Equation 10. We solve the remaining system of equations using the Conjugate Gradient method, giving us the final velocity field $\mathbf{u}$. See [21] for details.

4.2 Embedded Surface Mesh

In order to simulate a liquid, we also need to simulate the motion of the the visible surface at the fluid boundary. We choose to use an explicit triangle mesh for this fluid surface. The mesh is simply a collection of vertices connected by triangles. This surface mesh partitions space into two regions: a portion of space that is "inside" the simulated liquid, and a vast region of space that is "outside" of the liquid. Because of this clean partition, we use this surface mesh to mark the simulation domain in our fluid solver. In other words, the surface mesh tells the fluid simulation which cells should be simulated as fluid and which should not, depending on whether the cells lie inside or outside of the mesh surface. As such, we insist that this surface is manifold and closed, in order to have a clear definition of what regions are "inside" of the simulated fluid, and which regions are "outside."

4.2.1 Updating the Surface Mesh

Once we have solved for the velocity field $\mathbf{u}$ at the end of each time step, use it to move the fluid surface. This typically requires the solution of a partial differential equation (as in the level set method [112]), but we can break the problem down into many smaller, easier-to-solve ordinary differential equations.

Each surface vertex is implicitly associated with a velocity, because it has a unique position $\mathbf{x}_i$ and the velocity field is a function of space, $\mathbf{u}(\mathbf{x})$. In practice, $\mathbf{u}$ is

only defined at discrete regular intervals in our fluid grid, but we can use simple interpolation techniques to fit a smoothly-varying function to this data. In practice, we use tri-linear interpolation to find a velocity for a given position in space.

If we assume that no topological changes occur during the motion of the surface, we do not need to update any triangle connectivity; we simply need to solve the ordinary differential equation

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{u}_i \tag{11}$$

for each surface vertex i with position $\mathbf{x}_i$ and velocity $\mathbf{u}_i$. We can rearrange this equation to get the integral

$$\mathbf{x}_i^{\text{new}} = \mathbf{x}_i^{\text{old}} + \int \mathbf{u}_i dt \tag{12}$$

The simplest technique for performing this numerical integration for each surface vertex is to use the forward Euler method:

$$\mathbf{x}_i^{\text{new}} = \mathbf{x}_i^{\text{old}} + \mathbf{u}_i \Delta t \tag{13}$$

This forward Euler method works well enough for most purposes, however we choose to use a fourth order Runge Kutta method (RK4) [119] because it performed best in analytical tests and it is relatively inexpensive to compute.

Because we make extensive use of velocity interpolation when integrating the surface vertices, we need to be able to access the velocity field $\mathbf{u}$ in many regions just outside of the boundary of the fluid simulation. To make sure that our velocity field $\mathbf{u}$ adequately samples space, we extrapolate $\mathbf{u}$ outward using a fast marching method [21].

As we update the position of these surface vertices, the surface mesh will deform. If we do not change the connectivity of the triangle mesh, then triangles in the surface mesh can become severely distorted — some triangles may become much too large, while other triangles may have near-zero area. To address this problem, we

perform various "mesh-maintenance" operations at the end of each time step. We loop through each triangle in the mesh and perform changes if any triangle falls below some quality threshold. For example, if we find any edges that are too short, we perform edge collapses (deleting one of the vertices and two triangles). If we find any edges that are too long, we perform edge subdivisions (creating a new vertex and two new triangles). In addition, although we have not implemented this technique, it would be a good idea to implement edge-flipping operations in order to adjust triangle areas and aspect ratios without re-sampling the surface vertices.

Lastly, when we update the positions of the surface vertices, some vertices may lie within solid obstacles. One way to address this problem is to resolve the collision by projecting the vertices onto the surface of the obstacle. This works well in most cases, though it can cause problems when an entire thin sheet of liquid lies inside an obstacle. In such cases, simply projecting vertices onto the surface of the obstacle tends to create infinitely thin sheets of liquid. These degenerate cases can cause problems in later chapters, when the convex hull of the surface is completely planar. In practice, we used free-slip boundary conditions at these solid obstacles and ignored any resulting collisions. While this approach worked surprisingly well, a better approach to collision handling is desired.

4.2.2 Updating the Finite Difference Grid

As mentioned in the previous section, the fluid simulation uses the surface mesh to determine which cells will be active fluid cells, and which cells will be inactive "air" cells. This is essentially the same problem as *voxelizing* a surface mesh, or returning a set of grid cells that overlap the volume contained within a surface mesh. However, we can take this method a step further and actually compute a signed distance function in order to give ourselves some more information about the surface geometry. The scalar values of this signed distance function (negative values inside, and positive

values outside) are collocated with the cell-centered pressure values in the fluid grid.

We employ a voxelization-style method [98] to compute the signed distance function. We first voxelize the triangle mesh onto a grid by computing intersections with a rays in the x, y, and z directions. For each ray, we keep track of its inside/outside status with a counter. At the start of the ray (which is guaranteed to be outside of the mesh), the counter has a value of zero. For each intersection with the triangle mesh, we increment the counter if it intersects a triangle whose normal is facing the opposite direction of the ray, and we decrement it if it intersects a triangle with a normal facing the same direction of the ray. This way, all regions outside of the mesh will have a counter value of zero, regions inside the mesh will have a value of one, regions where multiple surfaces overlap each other will have a value greater than one, and regions that are inside-out will be negative. We store this counter value at each grid point to assign an inside/outside status, and then we compute the distance from the point to the nearest surface triangle in order to complete the signed distance calculation. Because our algorithms only need an accurate signed distance in the thin band surrounding the surface, we save time by only performing this distance calculation for grid cells that overlap surface geometry.

At this point, we can use this signed distance function to determine which fluid cells are inside of the surface (treating all "inside" cells as active fluid cells in the next time step). However, some regions of the surface may have been too thin to be adequately sampled by the fluid grid, so these thin regions would not be surrounded by any active fluid cells with this method.

We could use an iso-surface thickening technique to artificially inflate the zero-set of the signed distance function [75]. By uniformly subtracting a scalar equal to $\sqrt{3}$ times the width of a fluid cell, we can ensure that all thin features will be represented by negative values on the signed distance function grid. However, this strategy also activates additional fluid cells that do not need to be activated, like those just outside

of flat, planar regions. Although this method works well in practice, it adds additional mass to all fluid boundaries.

To avoid the allocation of additional fluid cells along the surface, we use an explicit geometric technique instead. We first find the set of all fluid cells that intersect or completely envelop any surface triangles. We then take the union of that set of cells and the original set of cells associated with negative values in the signed distance function. This final set of cells will adequately sample all thin features without adding unnecessary fluid cells in well-sampled areas.

4.3 Mass of a Fluid Cell

At this point, the embedded surface will not necessarily occupy the exact same volume as the simulation elements in general, similar to Section 3.4 of Chapter 3. This difference can cause inaccuracies in the simulation process, and there are several strategies we can take to limit these problems.

4.3.1 Uniform Mass

If we simply ignore the problem and assume that every fluid cell represents a unit of mass $\rho\Delta x^3$, where Δx is the width of a fluid cell, then the simulation will overestimate the total mass in the system. This is most noticeable in thin regions of liquid, when a thin surface sheet splashes into a larger body of water. The artificially large mass creates injects additional momentum into the system and creates a larger-than-expected splash. Although this behavior may actually be desirable in a special effects environment and more pleasing to watch, it is nevertheless inaccurate.

Errors in the uniform mass strategy are most visible when thin surface regions stretch out into even thinner surfaces. Here, the volume of these surface features is preserved, so the volume per unit length drops as the length increases. However, the mass per unit length stays the same with the uniform mass strategy, so a large amount of mass and momentum is added into the system whenever a small region

thins out.

Although the uniform mass method is the least accurate, it is also the simplest to implement. We used this method in our simulations, but we would like to implement one of the following more accurate methods in the near future.

4.3.2 Fractional Mass

One way to counteract these errors is to follow Section 3.4 of Chapter 3 and limit the mass based on the actual volume of liquid overlapping a fluid cell. This strategy is similar to that of Batty et al. [9] for accurate solid-fluid coupling within a single grid cell.

4.3.3 Ghost Fluid Method

A somewhat related idea is that of the ghost fluid method [45, 10]. This is a second-order approximation that essentially uses a linear extrapolation to assign pressure values outside of the free surface. Because the fluid pressure should drop to zero exactly at the surface, an extrapolation will give negative pressures outside of the fluid surface. We have not used this method for any of our experiments, but it has yielded high quality visual results in other research involving explicit surface meshes [25].

4.4 Surface Tension Approximation

Just like Section 3.5 of Chapter 3, we can approximate surface tension using our high resolution embedded surface mesh. To do this, we first compute the forces on the mesh in the same way as above, and we down-sample the forces onto the grid used for fluid simulation. To account for high resolution features that cannot be computed on the low-resolution fluid grid, we also apply Laplacian smoothing on the mesh surface. As noted by Brochu et al. [25], this type of simulation can quickly blow up if the high resolution features are not adequately removed by the smoothing procedure. I will

```
Calculate forces(viscosity,
                 gravity,
                 surface tension)
Calculate fluid velocity
If SurfaceTension
  Smooth the surface
Advect surface vertex positions
If Collision detection
  Update vertex positions
Subdivide large surface triangles
Collapse short surface edges
Compute signed distance function
Assign new fluid cells
```

Figure 14: Pseudo-code for one timestep of Eulerian fluid simulation

present a more stable high-resolution surface tension technique in Chapter 8.

4.5 Algorithm Overview

Figure 14 shows pseudo-code for a single time step of an Eulerian fluid simulation with a mesh-based embedded surface. The simulation first computes all of the Eulerian fluid forces (such as viscosity, gravity, and surface tension), then solves one time step of the incompressible Navier-Stokes equations to get a new fluid velocity for each fluid cell. If we wish to simulate surface tension, then we can apply geometric smoothing to the surface mesh (as explained in Section 4.4). A more accurate approach to surface tension will be discussed in Chapter 8. We next advect the surface vertices through the Eulerian fluid cell velocities using standard techniques for numerically integrating ordinary differential equations, and then adjust these vertex positions in the case of any collisions. To ensure that our surface mesh is adequately capturing surface details and making efficient use of memory, we subdivide long triangles and collapse triangles with short edges or small angles. Finally, we communicate the new surface configuration to the Eulerian fluid simulation by first computing a signed distance field from the surface mesh and then using it to designate the new active fluid cells.

Note that the timestep outline in Figure 14 assumes that our simulation has been properly initialized (we know the initial positions of the surface vertices, and we know which fluid cells are active in this time step). One simple way to ensure that we have these initial conditions is to use an input triangle mesh to determine the initial surface configuration, compute a new signed distance function from this mesh, and then assign new fluid cells from the signed distance function. After initializing the mesh and fluid cells in this way, we have all the information we need to start looping through time steps (like Figure 14) until the simulation terminates.

4.6 Discussion and Limitations

This chapter discusses a straightforward method for coupling a grid-based Eulerian fluid simulation to a mesh-based Lagrangian surface. As such, it inherits the benefits of both the Eulerian and Lagrangian simulation techniques. By using a semi-Lagrangian advection scheme for the Eulerian finite difference scheme, we can guarantee that our fluid advection will be unconditionally stable for large time steps. The Lagrangian nature of our surface mesh also avoids the accumulation of re-sampling errors that are common in Eulerian surface tracking schemes.

The method discussed in this chapter also inherits some weaknesses from the coupled Eulerian and Lagrangian schemes. For example, it is difficult to calculate Eulerian surface tension forces from a Lagrangian surface mesh if the surface has a higher spatial resolution than the fluid simulation grid (as pointed out by [25]). The geometric smoothing advocated in Section 4.4 does a great deal to alleviate this problem by reducing the spatial frequencies of the surface mesh, but this artificial smoothing of surface details is far from an ideal solution. A more thorough and effective solution will be presented in Chapter 8.

One major problem with this approach as presented so far is that it does nothing to address topology changes in the surface geometry. Lively fluid animations can have

drastic changes in the connectivity of surface geometry, so it is important that we address this issue. Chapters 5, 6, and 7 will discuss this problem and present some solutions in more detail.

CHAPTER V

TOPOLOGY CHANGES IN DYNAMIC SURFACES

This dissertation is concerned with surfaces that evolve through time. As these surfaces evolve, they not only change shape and move around, but they can also merge together, split apart, appear, or disappear (Although this dissertation may not be as concerned with surfaces appearing and disappearing, these behaviors are still possible for arbitrarily deforming generic surfaces). Such events are called changes in the surface's *topology*, and this chapter will discuss the nature of these topological changes.

To learn more about the behavior of a surface through time, we can plot the its position at every point in space and in time on the same graph. In Figure 15, we visualize a one-dimensional surface swept through time, with its position in space represented on the horizontal axis and time on the vertical axis. This time-swept figure creates a polygon in space-time. A vertical line in this graph represents a fixed point in space, and horizontal line represents a fixed instant in time. Whenever a topological change occurs, parts of the surface will either appear, disappear, split apart, or merge together. These instances can be identified as critical points in space-time. In Figure 15, a dashed line is drawn at each instant a topological change occurs. Line **A** marks the global minimum in time, signifying the initial creation of the surface. As time advances, the surface gets wider (occupies more space) until it splits into two pieces at instant **B**. In the period of time between **B** and **C**, the two new surfaces drift away from each other, as the right surface inflates. Instant **C** represents another local minimum, as the right surface splits apart once again. Line **D** marks our first local maximum, as the leftmost two surfaces merge together.

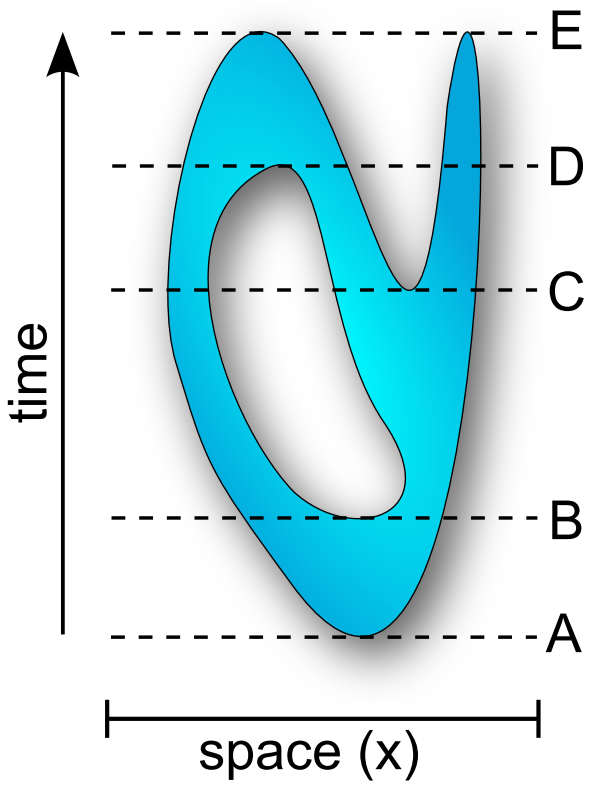


Figure 15: This figure shows a one-dimensional dynamic surface swept through time. Topological changes to a dynamic surfaces can be represented as critical points in space-time, and these critical moments are marked by dashed horizontal lines.

Finally, the global maximum is found at time **E**, where the remaining surfaces shrink until they disappear.

5.1 Necessity of Topology Changes in Simulated Physics

In order to faithfully and simulate interesting natural phenomena, we must allow our dynamic surfaces to change their topology. These topological changes are not only necessary to capture realistic physical effects, but they are also a useful tool for simplifying overwhelmingly complicated systems for the purposes of computer animation.

5.1.1 Importance in Nature

Topological changes occur naturally in several settings. If we are concerned with the dynamics of fluids and liquid surfaces, then topological changes occur whenever the surface breaks into droplets, or whenever different surfaces merge together upon contact. Cohesive forces between liquid molecules cause surfaces to merge together the instant they touch each other, and surface tension forces cause surfaces to rip apart whenever they form thin tendrils (this phenomena is known as a Rayleigh-Plateau instability, and it is primarily responsible for the degeneration of a splashing surface of water into a mist filled with tiny droplets). If our simulations do not allow topological changes such as merging, then surfaces will build up several layers of water separated by nothing but a vacuum. Because strong surface tension forces may act on these layers, even though natural topological changes would have deleted these liquid layers the instant they came contact with one another, the resulting forces computed in this system will be ultimately incorrect. Similarly, if we do not allow surfaces to tear apart, then surface tension effects will routinely create infinitely thin tendrils of liquid. Aside from this type of behavior being highly unnatural, it can be especially problematic for the numerical stability of our simulations.

If, instead of animating liquid surfaces, we are concerned with simulating elastic or viscoelastic materials, then we must account for topological changes whenever substances stick together or fracture. If we are not careful about merging surfaces together, then our simulations may create complicated intersecting surfaces instead of properly joining together materials. Similarly, materials that are never allowed to fracture will behave unnaturally when confronted with large forces, and they may be stretched infinitely thin without ever snapping apart.

5.1.2 Importance in Animation

Topological changes like surfaces splitting and merging are not only important for recreating natural phenomena in our simulations. They are also important for ensuring that our simulations have a manageable workload. In particular, if we neglect to merge together any surfaces, then only a few seconds of a dynamic splashing liquid will generate an overwhelming number of folded surfaces that should have naturally merged together. Aside from this behavior being unnatural, the computational load necessary to store and operate on each of these surface primitives will skyrocket. As mentioned in the previous section, we can also run into numerical problems if we allow surfaces to become arbitrarily thin without imposing topological changes. This results from the inevitable division by very small numbers as the angles and edge lengths of surface triangles shrink. In order to allow for fast and stable computer animations of physical phenomena with dynamic surfaces, we must properly handle their topological changes.

5.1.3 Dynamic Topology Changes in Practice

In practice, because we use a triangle mesh that evolves through time as our dynamic surface, we are interested in methods for explicitly enforcing topological changes on a triangle mesh. Unfortunately, the solution is not straightforward. Many arbitrarily complicated shape configurations can arise, and our code must handle every one of them robustly. It is not always as simple as identifying two surfaces that are close to each other and then sewing them together — self-intersecting surfaces can form extremely complex shapes. We must also guarantee that the surface is closed, manifold, and consistently oriented after every one of our topological operations — otherwise, many important assumptions will be violated and the physics simulation will fail.

We have made progress on this problem, and I will discuss two possible solutions

in the following chapters. In Chapter 6, I will explain how to reduce this complex problem into a series of problems whose solution is known — voxelization, isosurface extraction, and a special limited set of mesh surgery operations involving edge collapses and triangle subdivisions. This method was one of the first methods in computer graphics for robustly tracking a dynamic surface mesh in the presence of topological changes. Though its physical behavior of the surface is only accurate if it can be adequately resolved on a voxel grid. In Chapter 7, I will discuss a more versatile method for liquid topological changes, inspired by the behavior of liquid surfaces in nature. This newer method allows much higher resolution liquid surfaces which behave in a physically plausible manner even when under-resolved by the fluid simulation, and it implies several exciting avenues of future work.

CHAPTER VI

MATCHING THE TOPOLOGY OF A VOXELIZED IMPLICIT SURFACE

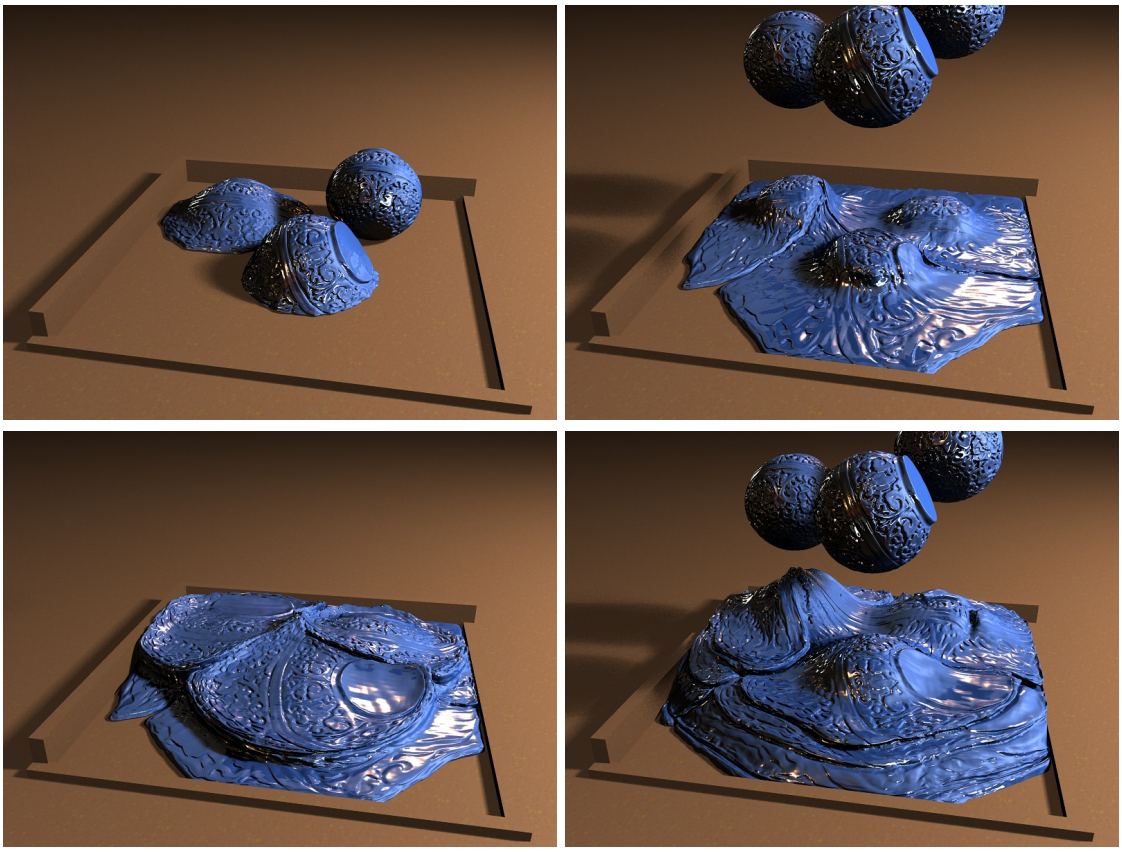


Figure 16: Dropping viscoelastic balls in an Eulerian fluid simulation. Invisible geometry is quickly deleted, while the visible surfaces retain their details even after translating through the air and splashing on the ground.

In this chapter, we introduce our first method for handling topological changes with a deforming triangle mesh [154]. This method first uses the triangle mesh to create a signed distance function on a regularly sampled voxel grid. This signed distance function represents an implicit surface at its zero-levelset. This implicit

surface will not be exactly the same as the explicit surface from which it was derived. The implicit representation is limited by its sampling resolution, so the topologies of the implicit and explicit surfaces may differ where the signed distance function is under-sampled in contrast to the features of the surface mesh.

As a result of these restrictions, the implicit surface will have a simpler and cleaner topology than the explicit surface. Indeed, we use the implicit surface to indicate where topological changes may be necessary in the explicit surface by identifying regions of space where the topologies of the two surfaces differ. Furthermore, we use the implicit surface to actually clean up topological problems in the explicit surface. Because the implicit surface is guaranteed to have clean topology everywhere, we can clean up the topology of the explicit surface by locally replacing patches of its surface with new surface patches from the implicit surface.

6.1 Overview of Approach

This algorithm for topological changes to a triangle mesh can be folded into the surface tracking algorithms mentioned in Chapters 3 and 4. The main steps of our surface tracking algorithm are shown in Figure 17. The tracker is given a closed manifold mesh M that represents the surface of the given material. The surface tracker first moves the vertices of this mesh according to the update rule given by the simulator: typically via interpolation of simulation nodes or numerical integration through a velocity field, as described in Chapters 3 and 4, respectively. Next, the space around the mesh is finely sampled on a regular grid, and the result of this sampling procedure is a signed distance field D. The sign of a sample in the distance field is determined by whether the sample is inside or outside of the mesh M, and the magnitude is the distance to the nearest part of the mesh. This distance field is then traversed, and the eight distance samples at the corners of each cell are examined to determine whether a topological event may be taking place at the given cell. There

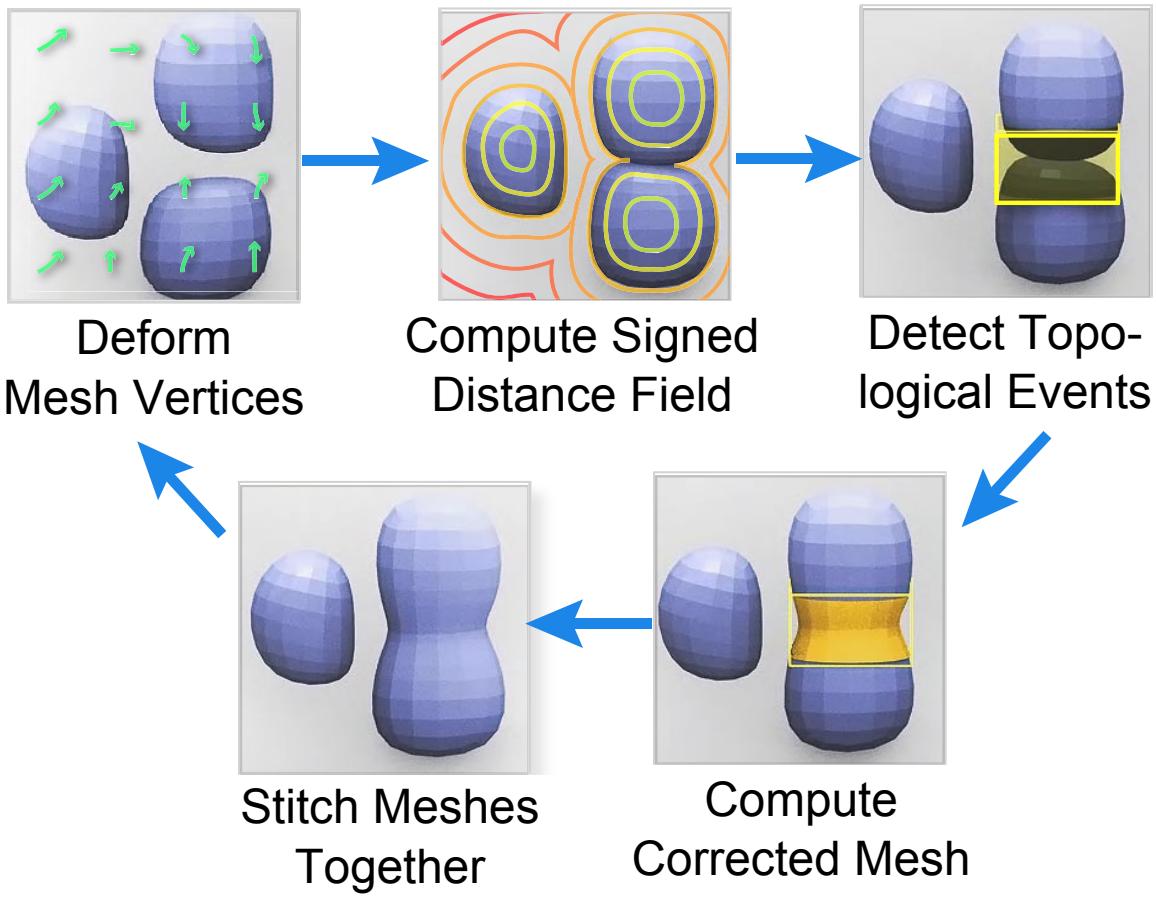


Figure 17: Overview of the topology modification pipeline, beginning in the upper left by deforming the input mesh. This method handles topological changes by comparing an explicit surface to a similar implicit surface. The explicit surface is given as input, and the implicit surface is created from the signed distance function.

are several possible decision processes that may be used, and one that we favor is the *deep cell test*, which will be explained in Section 6.2. This results in a list of cells that are marked for modification. We clip the surface mesh M to the boundary of this collection of marked cells, retaining only the portions of the mesh that are outside of the marked cells. This will be described in Section 6.3. We then form new mesh components inside the marked cells using marching cubes, and the clipped exterior mesh is joined to the interior mesh components. The result is a new mesh that has been modified in regions of topological change. We give details about each of these steps in the sections that follow.

6.2 Detection of Topological Events

Assuming we have already updated our surface mesh, we then decide whether we should split any surfaces apart or merge them together. We first calculate a signed distance field D from our surface mesh, and then we examine the structure of this field to help decide where any topological events should take place.

6.2.1 Signed Distance Field Calculation

We choose to place our signed distance function on a regular grid that encloses our surface mesh M. Note that the only places where topological changes can occur are at grid cells that intersect M (the surface cells). Thus we calculate the distance to the triangle mesh at each grid point that touches a surface cell by calculating the exact distance to each nearby triangle and taking the minimum. We then use the voxelization method described in Section 4.2.2 to determine which grid points lie inside of M, and which lie outside. We assign a positive signed distance to the grid points outside of M and a negative sign to the points inside of M. Because we only calculate the signed distance at grid cells touching the surface, and because we only sample the nearest triangles for each distance query, this calculation of D is efficient.

6.2.2 Complex Cell Test

At this point, we have two surface representations: an explicit surface mesh M, and a signed distance function D that implicitly defines a surface at a specified grid resolution. We can contrast these two surface representations to give us an idea of where the surface is topologically complex.

One way to do this comparison is using *digital topology* [123]. Although digital topology is a powerful tool for detecting topological events, it is inappropriate for our problem, because it requires a strict CFL condition for the movement of the surface. In addition, digital topology will only detect actual topology changes, leaving alone homeomorphic transformations. We would like to simplify our surfaces in invisible

regions, such as large folds, even though this is not technically a topological change.

Another method we could use for detecting topology changes involves *complex cell tests*. Complex cell methods essentially decide whether our surface M is too complicated to represent with a piecewise linear isosurface extracted from D. Varadhan *et al.* [149] use a complex cell test in addition to a star-shaped test to decide whether their isosurfaces are topologically equivalent to an input mesh. Our method for detecting topological events uses a similar test, but we tailor ours to minimize re-sampling of the visible surface.

We can think of D as a network of cubic cells with edges connecting each grid point to its six closest neighbors. In brief, a cell in D is complex if the marching cubes algorithm [88] will produce a significantly different surface from M in that cell. More specifically,

- A *complex edge* is an edge in D that intersects M more than once.
- A *complex face* is a square face in D that intersects M in the shape of a closed loop or touches a complex edge.
- A *complex cell* is a cubic cell in D that has any complex edges or complex faces, or any cell that has the same sign of D at all of its corners while also having explicit geometry from M embedded inside of it. The act of testing a cell for complexity is called the *complex cell test.*

This complex cell test can be used to mark a region in space where any topological changes occur in our surface. However, a straightforward application of this test will also mark detailed surfaces as topologically complex (see Figure 18). If we wish to preserve surface details like sharp corners, we must significantly modify this test.

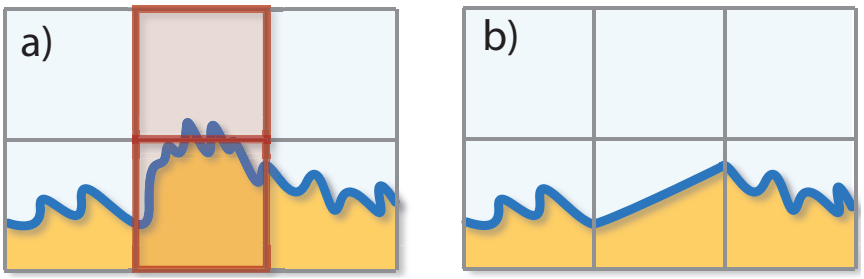


Figure 18: This two-dimensional example shows how the complex cell test will aggressively re-sample detailed surface features, even in the absence of topological changes. In (a), we show cells that the complex cell test will mark due to multiple edge intersections, and we show the re-sampled surface in (b).

6.2.3 Deep Cell Test

Because we are primarily interested in the fidelity of the visual surface, we would prefer that most surface re-sampling occurs only in invisible regions or in the presence of major topological changes. To avoid the excessive re-sampling of highly detailed surfaces, we do not wish to mark all complex cells as topological events. Instead, we start by contrasting the interface of our explicit surface M with the interface of our implicit surface D. In the interest of surface detail preservation, we ignore any complex cells that are sufficiently close to both the interfaces. We only mark a complex cell that is at least one cell away from the isosurface interface, as illustrated in Figure 19. This *deep cell test* is necessary because subtle bumps in surface geometry can still trigger any complex cell test, but only cells with geometry fluctuations larger than a cell length will be marked by our deep cell test.

6.2.4 Self-Intersection Tests

In addition to topological changes triggered by surface proximity, we also choose to mark cells that indicate significantly large self-intersections in M. This test is performed while marching rays through the mesh in the voxelization phase of our signed distance calculation. Whenever a ray intersects a triangle, we determine whether the ray is entering or leaving the surface by calculating the dot product of the ray

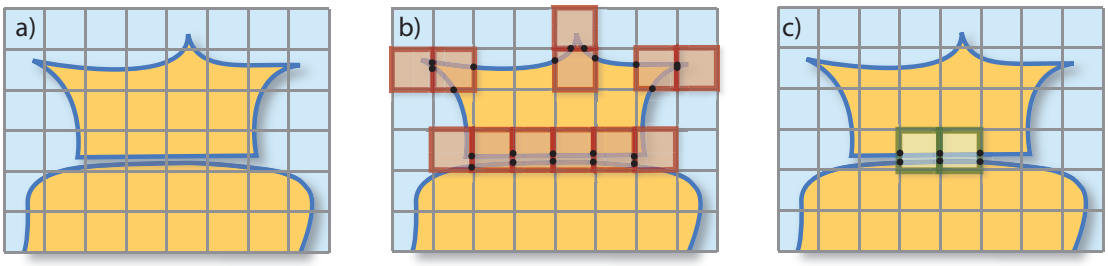


Figure 19: A two-dimensional illustration of our deep cell test. Figure (a) shows an input surface mesh M (dark blue line) with a visualization of the corresponding signed distance field D, where orange points are inside of the surface, and light blue points are outside. Next is a figure showing all complex cells (b). The rightmost figure shows all deep cells (c). Note that the deep cells only label geometry necessary for a topological change, while the complex cells aggressively label important surface details.

direction with the triangle normal: a negative dot product indicates that the ray is entering the surface, while a positive dot product indicates an exit ray. In our implementation, we initialize an integer variable to zero at the start of the ray march, and then we increment the value if the ray enters the surface and decrement the value if the ray leaves. As the ray passes through each grid node, every node that does not have an integer value of zero or one necessarily is in a region of multiple overlapping or inside-out surfaces. We mark any cells touching these grid nodes as topologically complex. To ensure that this check is robust at triangle edges and vertices, we can use any techniques from the large literature for calculating robust ray-triangle intersections. We obtained robust behavior by simply jittering any vertices that lie too close to a ray.

6.2.5 Additional Topological Controls

The algorithm up to this point allows a great deal of control over which cells should be marked and re-sampled. For example, a simple method for preserving thin features is to only allow merging, but avoid splitting. This is done by not marking deep cells with corners that lie completely outside of the mesh. There are only two types of deep cells (those that are completely inside the implicit surface, and those that are

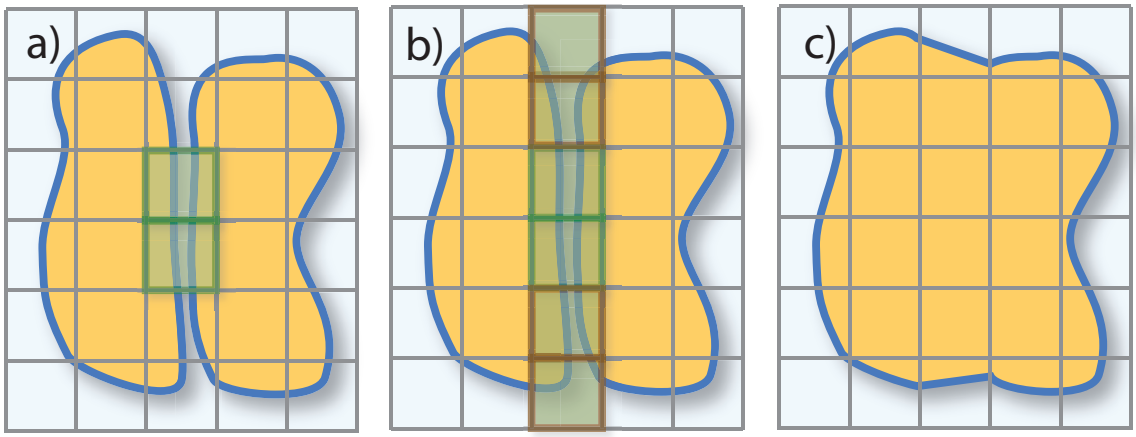


Figure 20: A two-dimensional illustration of the cell-marching step. In (a), all deep cells are marked. We march outward along complex edges and faces until the marked region is topologically simple (b). The right figure (c) shows a topological change after re-sampling the marked cells.

completely outside), and each type corresponds to a merging or splitting event. By never marking deep cells that are outside of the mesh, we can avoid splitting events and preserve thin features like sheets and spikes. This strategy of never marking deep cells outside of the mesh works well up to a point (see Figure 29), but it will cause problems when thin regions collide with regions that we want to re-sample. Re-sampling these thin regions will lead to serious popping artifacts, and we offer a solution to this problem in Chapter 7.

Once we have marked every cell that we wish to re-sample, we want to replace M in these cells with a topologically-simple isosurface extraction. However, we need to ensure that every cell on the boundary of our marked cell region provides a topologically simple interface with marching cubes. In other words, the boundary must contain no complex edges and no complex faces. We execute a flood fill algorithm, starting with the initially marked cells and marching outward across complex faces, until the entire region of marked cells is bound by topologically simple cube faces (see Figure 20). Once this region has a clean interface with marching cubes, we can perform surgery on the mesh.

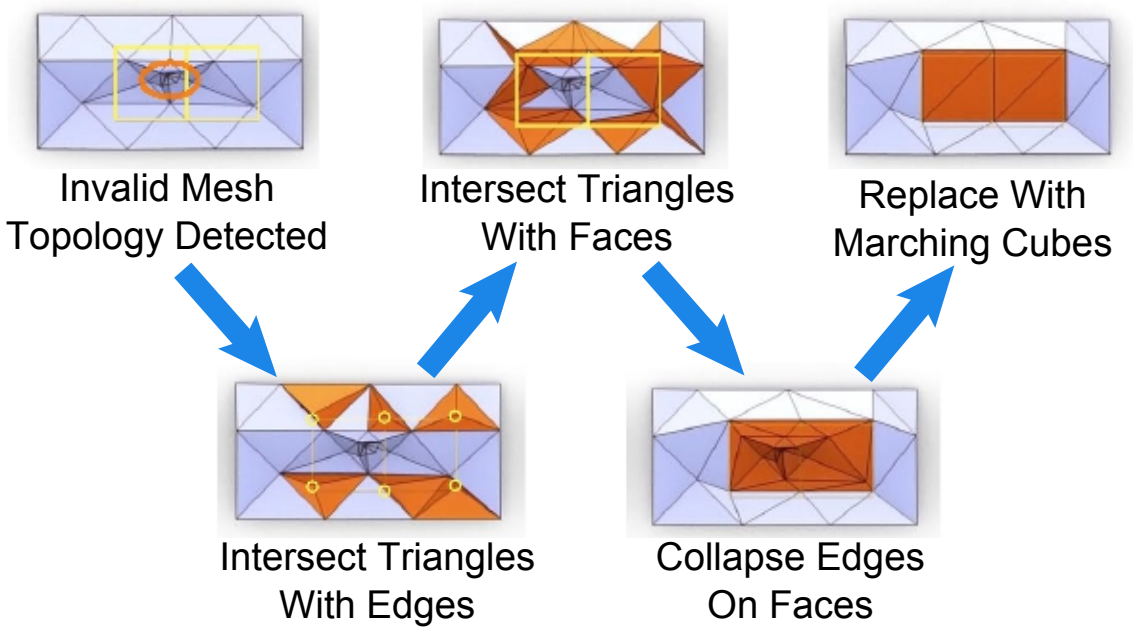


Figure 21: The steps involved in locally replacing the mesh with a topologically simplified piece that is generated using Marching Cubes.

6.3 Altering the Mesh Topology

At this stage in the algorithm, we have a mesh M, a distance field D, and a list of marked grid cells. Each of these marked cells describes a region of space in which we will locally remove the topologically complex explicit surface M and replace it with the topologically simple extracted isosurface of D. The surface inside of these cells is computed with a marching cubes algorithm and connected to the original mesh at the cell boundaries. To maintain a manifold surface mesh, it is important to ensure that the interface between the isosurface and the explicit surface matches up perfectly. We spend the rest of this section explaining how to compute this interface efficiently and robustly.

6.3.1 Sewing Meshes Together

Here, we describe the basic method for matching the interface between an explicit triangle mesh and an extracted isosurface. We also explain this algorithm graphically in Figure 21.

1. After marking topologically complex cells, we find each triangle that intersects a cell edge on the boundary of the marked region. We then calculate the intersection point between the triangle and edge, and we subdivide the triangle into three new triangles that share a vertex at the intersection point. We call this newly created vertex a *type 1* vertex.

2. Next, we find all triangle edges that intersect a cell face on the boundary of the marked region. We split each triangle edge at the point where it intersects the face, subdividing the two original triangles into four and inserting a new vertex on the face. We call this face vertex a *type 2* vertex.

3. At this point, the triangle edges between type 1 and type 2 vertices describe a single piecewise linear curve connecting type 1 vertices along each boundary cell face. We want to simplify this curve until it is a straight line between type 1 vertices. We repeatedly collapse the edges between type 1 and type 2 vertices until all type 2 vertices are removed from the mesh.

4. After all subdivisions have been performed, no triangles will cross the faces of any marked cells. That is, each triangle will lie completely inside or completely outside of the marked region. We delete all triangles that lie completely inside of the marked region.

5. After all edge collapses have been performed, at most one vertex lies on each boundary edge, and the mesh intersects the marked cell faces along a single line segment in each boundary cell face – the boundary faces are topologically simple and line up perfectly with a piecewise linear isosurface extraction of the signed distance field D. We use marching cubes to generate a triangle mesh in the marked region, and we connect the meshes together at the type 1 vertices.

Our method of modifying the topology of a mesh is similar to that of Du et al. [42], but with a couple of significant differences. First, Du et al. only perform re-meshing when they detect a mesh self-intersection, whereas we also use this surface re-sampling for the more general purpose of permitting meshes to merge and split. Second, their approach forces their groups of marked cells to be in rectangular blocks, which means they often have to mark a much larger portion of the mesh for modification. We allow our marked cells to be more sparsely distributed, such as a marked set of cells whose union forms concave regions like the shape of the letter L. Allowing these more general marked cell configurations necessitates extra checks during stitching, and we describe this next in Section 6.3.2.

6.3.2 Robustness

For robust simulation and intersection tests, we must ensure that our triangles are well-shaped. We adaptively subdivide surface triangle edges when they become too long, and we collapse edges when they become too short or when their triangles have bad aspect ratios. We also perform edge flips after the creation of type 1 vertices if any newly created triangle has particularly small angles. We avoid any edge-flip and edge-collapse operations that will create nonmanifold geometry. The edge-collapse operations may cause some re-sampling, but only in very small features. The triangle subdivision does not remove any surface details.

We also must take care to maintain simple topology along boundary faces before sewing the meshes together. Specifically, performing an edge decimation in step 3 of the above algorithm can deliver collateral damage to the surrounding cells by creating an additional curve between type 1 vertices on the face of a different boundary cell. If these edges are collapsed in the wrong order, we can easily force ourselves into creating nonmanifold geometry. For this reason, we delay an edge collapse operation if it will prematurely connect other type 1 vertices. Typically, collapsing other edges

first will quickly resolve this degenerate case.

Unfortunately, on rare occasions we can arrive at a gridlock situation in which no safe edge collapses can be performed, and we cannot produce a clean interface to the isosurface. In such situations, we simply add the surrounding cells to the existing list of marked cells and repeat the flood fill steps in Section 6.2.3, replacing the nonmanifold geometry with the topologically simple isosurface.

6.4 Integration with Physics

For integration with physics animation systems, any changes to surface topology must be communicated to the physics calculations. We have integrated our surface tracker into both a finite element viscoelasticity solver and an Eulerian fluid solver. To couple the surface tracker with an embedded mesh FEM solver, the finite element mesh should be recomputed after any major topological changes. For this reason, we only calculated topological changes immediately before re-mesh events in the simulator. Coupling with an Eulerian fluid simulator is similar. We use the fluid velocity field to move the surface, and we use the surface to update the active grid cells in the simulation (the actual location of the fluid). We chose to only calculate topological changes once per frame of the output animation.

6.5 Input Parameters

The algorithm in this chapter has relatively few parameters to set. First, the underlying physics simulation method has many physical parameters to set. Section 3.7 discusses the important parameters if a Lagrangian FEM simulation is used. If an Eulerian fluid simulation is used, then there are several parameters to consider, including viscosity, surface tension strength, grid size, and solid boundary conditions. We prefer to use free-slip boundary conditions in our simulations.

The topological algorithm mainly deals with parameters involving the surface mesh and the topological grid. Important parameters to consider here are the edge

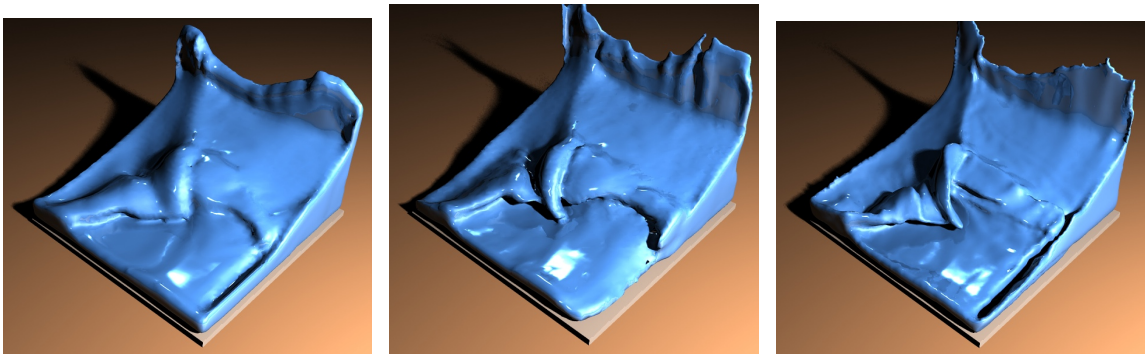


Figure 22: Comparison between different surface tracking methods. Left: Level set. Middle: Particle level set. Right: Our mesh-based tracker.

length in the topological grid, the minimum surface edge length, the maximum surface edge length, and factors concerning triangle area and shape, like the minimum angle or the ratio between the longest and shortest edge.

6.6 Results

In this section we describe the results of our surface tracking method using two different simulators and a variety of materials properties.

We compare our surface tracker to two other popular surface tracking methods in Figure 22. This figure shows how three different surface trackers perform on a dam-break fluid example. The left portion of the figure shows the result of using a pure level set approach with MacCormack advection [127]. Note that most of the surface details are washed away by this method. The middle image shows the result of the particle level set approach (note that we do not explicitly render the escaped particles). Here, more of the surface details are kept. The right image shows the result of our surface tracking method. Note the detailed features that are visible near the middle of this image where a wave has folded over onto a flat region of the liquid. The same Eulerian grid fluid solver and the same computational grid resolution of 64^3 was used for all three simulations.

Figures 23 and 16 show examples of dropping complex geometry onto a flat surface.

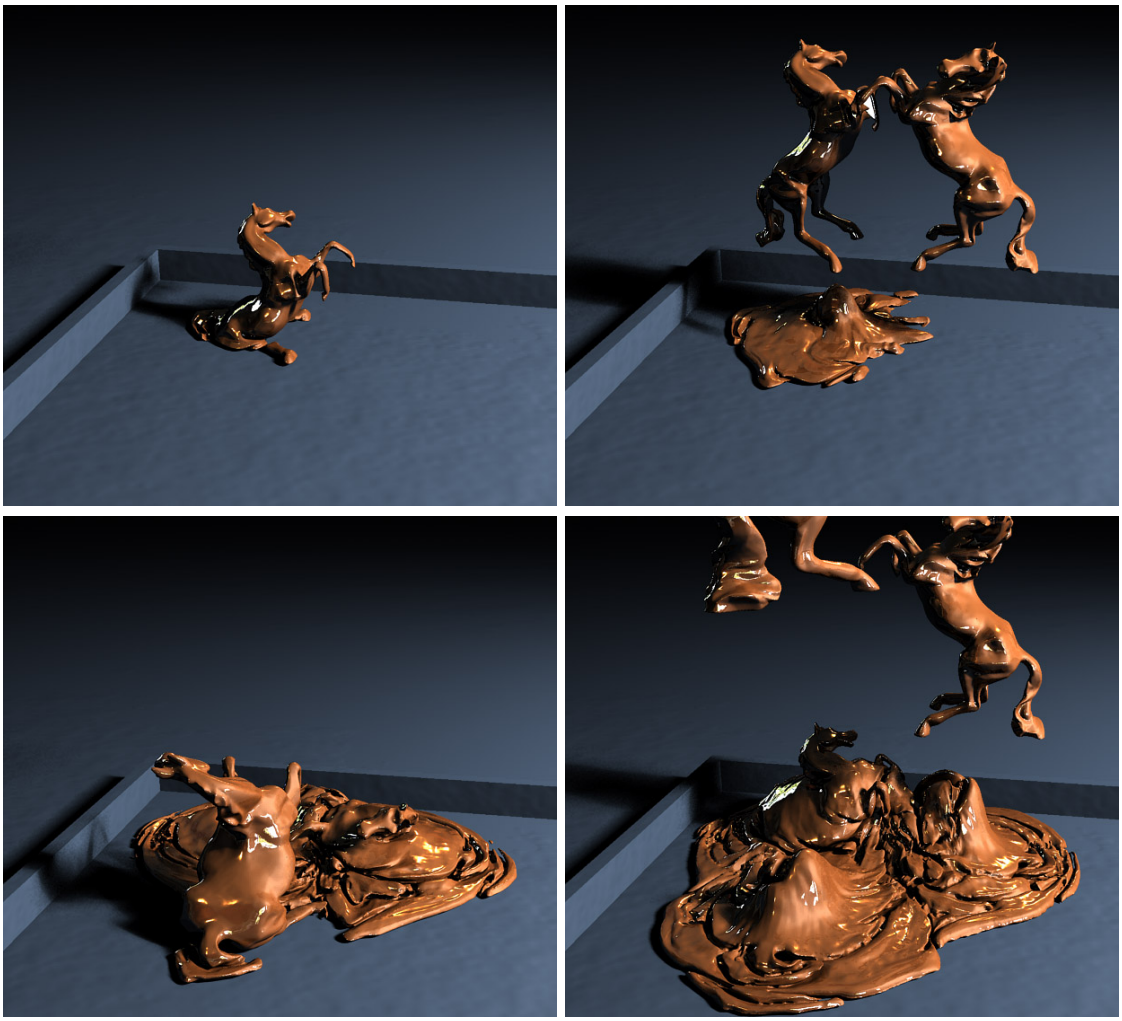


Figure 23: These images from an animation show viscoelastic horses being dropped onto one another. Many topological merges occur, yet details of the surfaces are kept.

In Figure 23, the objects that are being dropped are model horses. The underlying visco-elastic simulation [55] had a resolution of $60 \times 60 \times 90$ cells in both cases, while the marching cubes reconstruction was run with a three times higher resolution. The thin legs of the horses would be difficult to resolve using only a distance field representation. Figure 16 shows several ornately carved spheres in free-fall. The details that are carved onto the surface of these models are retained by our method because regions of the surface are not re-sampled until a topological event is triggered, and most areas of the visible surface are never re-sampled at all. Note the detailed folds that are created and retained when the surfaces push into one another. Each

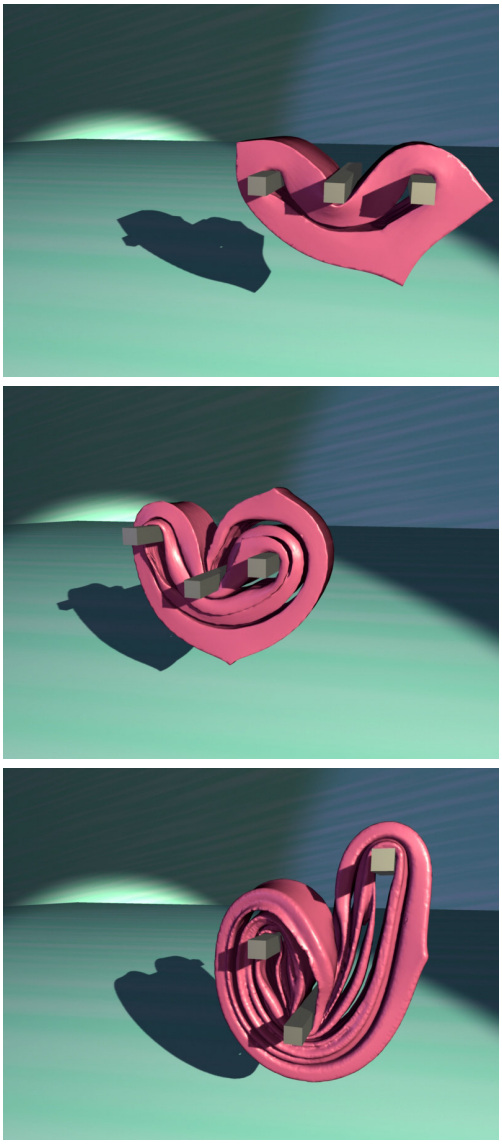


Figure 24: A virtual taffy-pulling machine creates complicated surface folds.

sphere initially has more than 420,000 triangles. In total, 15 spheres are dropped. Due to the merging of sheets, the final surface has roughly 1.6 million triangles. The simulation of this example took 63 seconds on average per frame.

Figure 24 shows a taffy pulling device. The initial torus of soft material is stretched and folded by three bars. In the absence of topological merging, each cycle of this mechanism would double the size of the surface. Our simulation merges the folded portions of the surface, which has the consequence of keeping the size of the surface

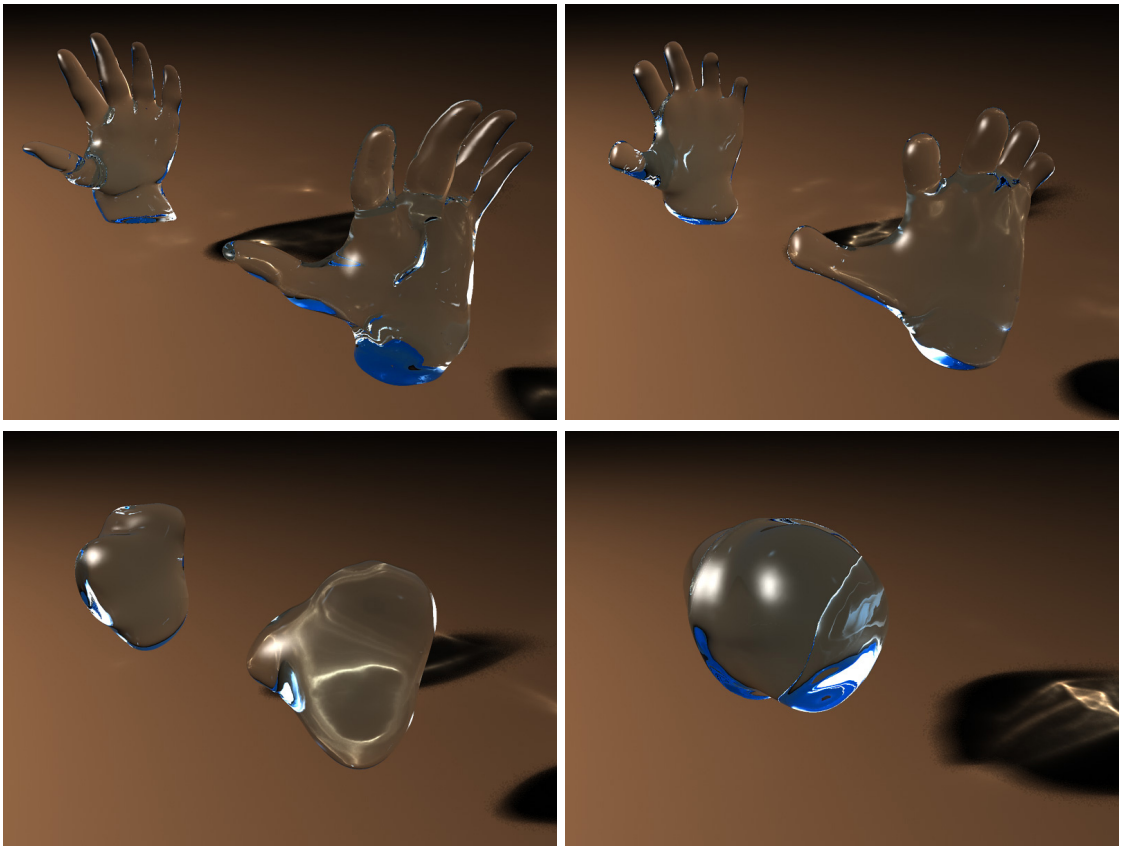


Figure 25: Clapping hands influenced by strong surface tension forces.

tractable. Moreover, the surface of our taffy retains geometric details from the folding process that would be washed out by a level set surface tracker. The physical simulation for this example was performed using our FEM simulator.

Figure 25 shows how we can use a mesh-based surface tension technique [24, 156], effectively computing high resolution surface tension forces in an Eulerian fluid simulator. This particular example illustrates how topological changes significantly affect simulated phenomena. Two liquid hands are heavily influenced by surface tension effects, first forming individual droplets and then merging together. Without the topological change, these two droplets would simply bounce off each other. Figure 29 shows frames from an animation of several plastic blocks dropped onto a plane. The blocks immediately merge and then splash outward, creating a thin sheet. By the end of the simulation, this sheet is far thinner than the topological grid resolution,

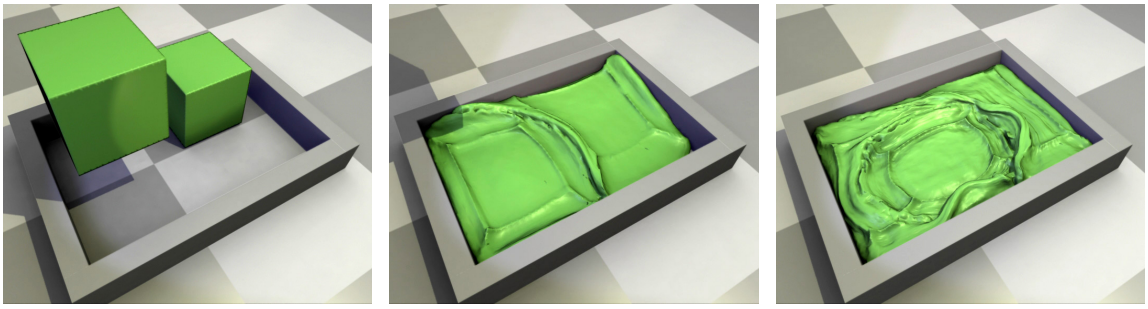


Figure 26: Three viscoelastic cubes merge as they drop into a pool.

but we prevented its deletion with the simple method for topological control described in Section 6.2.3. Our final example shows how our method can split a surface. Figure 28 features a stiff plastic cow being stretched out and then ripped into pieces by two passing blocks.

For a direct comparison between our surface tracking and other state-of-the-art methods, we performed tests with analytical velocity fields using implementations of a level set, particle level set, semi-Lagrangian contouring, and our method. Figure 27 shows how our method performs in the "Zalesak's Sphere" example as defined by Enright et al. [46]. Note that the sharp corners are completely preserved because of our deep cell test.

Out of the total time spent on our surface tracking algorithm, the core topology handling takes approximately 60% of the computation. The calculation of the signed distance field requires about 20%, while the other steps, such as the mesh update, the mesh subdivision, and interfacing with the simulation code take the last 20%. The ratio between computations for surface tracking and simulation strongly depends on the complexity of each part. For the example of Figure 16, with a very complex surface mesh and a coarse simulation, the surface handling takes up the majority of the computations. For a more complex simulation, such as the $200 \times 100 \times 100$ fluid simulation of Figure 25, the fluid simulation requires roughly 2/3 of the total runtime. In this case, the overall computations took 12.9 seconds per frame on average, while

for a smaller example, such as the 64^3 simulation of Figure 22, each frame took on average slightly less than one second.

The Lagrangian FEM simulations, on the other hand, featured far fewer topological changes, so computation times were dominated by the physics simulation. The

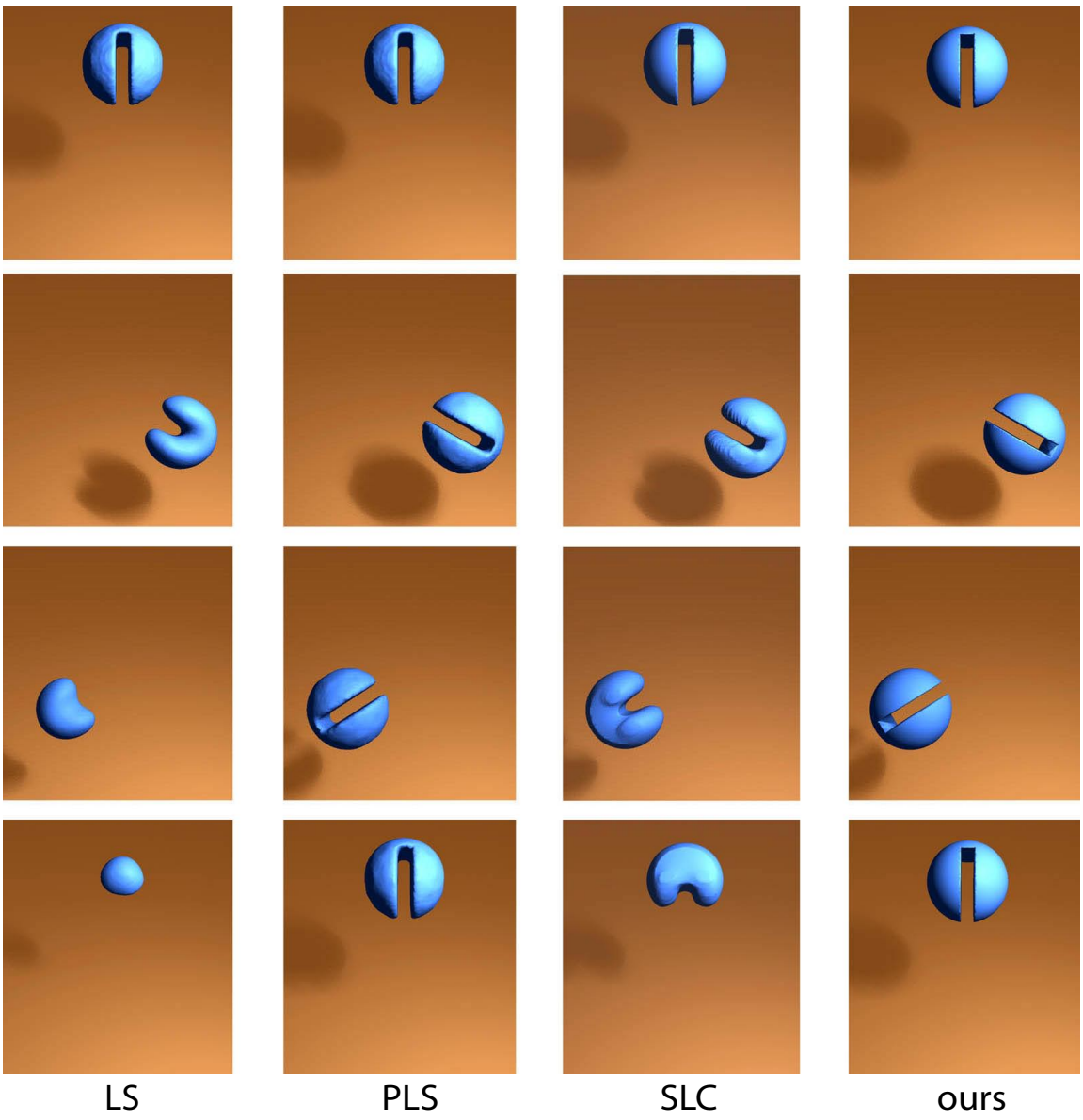


Figure 27: Results from the Zalesak Sphere example for implementations of a level set (far left), particle level set (middle left), semi-Lagrangian contouring (middle right), and our method (far right).

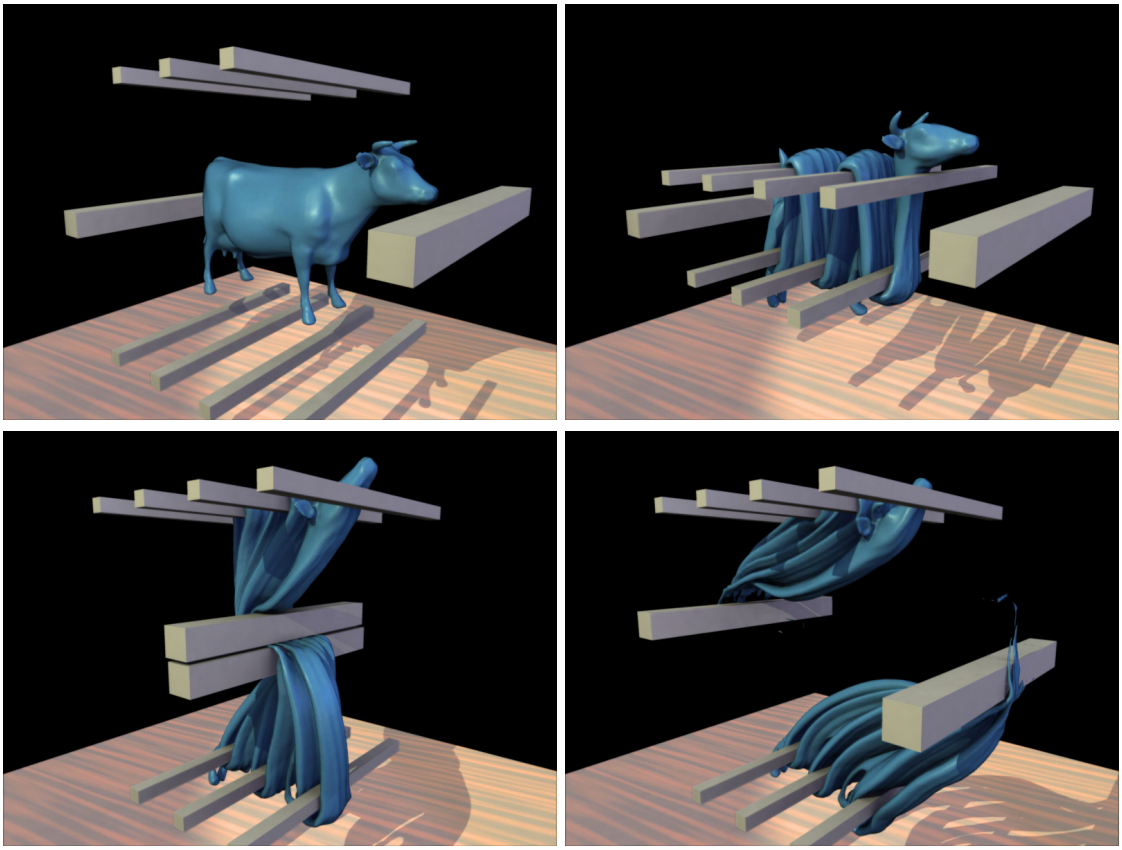


Figure 28: A stretched cow that is torn when two bars scissor together.

cow animation in Figure 28 finished with 12,000 elements and 250,000 surface triangles, the four cubes simulation in Figure 29 finished with 105,000 elements and 206,000 surface triangles, and the simulation of three cubes at the start of our video finished with 24,000 elements and 45,000 surface triangles. Each of these simulations took about 3 minutes per frame and spent less than 1% of the time on topological detection and meshing code.

6.7 Discussion and Limitations

There are several limitations to our approach. Chief among these is that the method identifies topological events and locally re-meshes based on the particular cell size of the distance field. Changes to the mesh can be noticeable if this cell size is too large. In addition, our method's topological classification will ignore small features

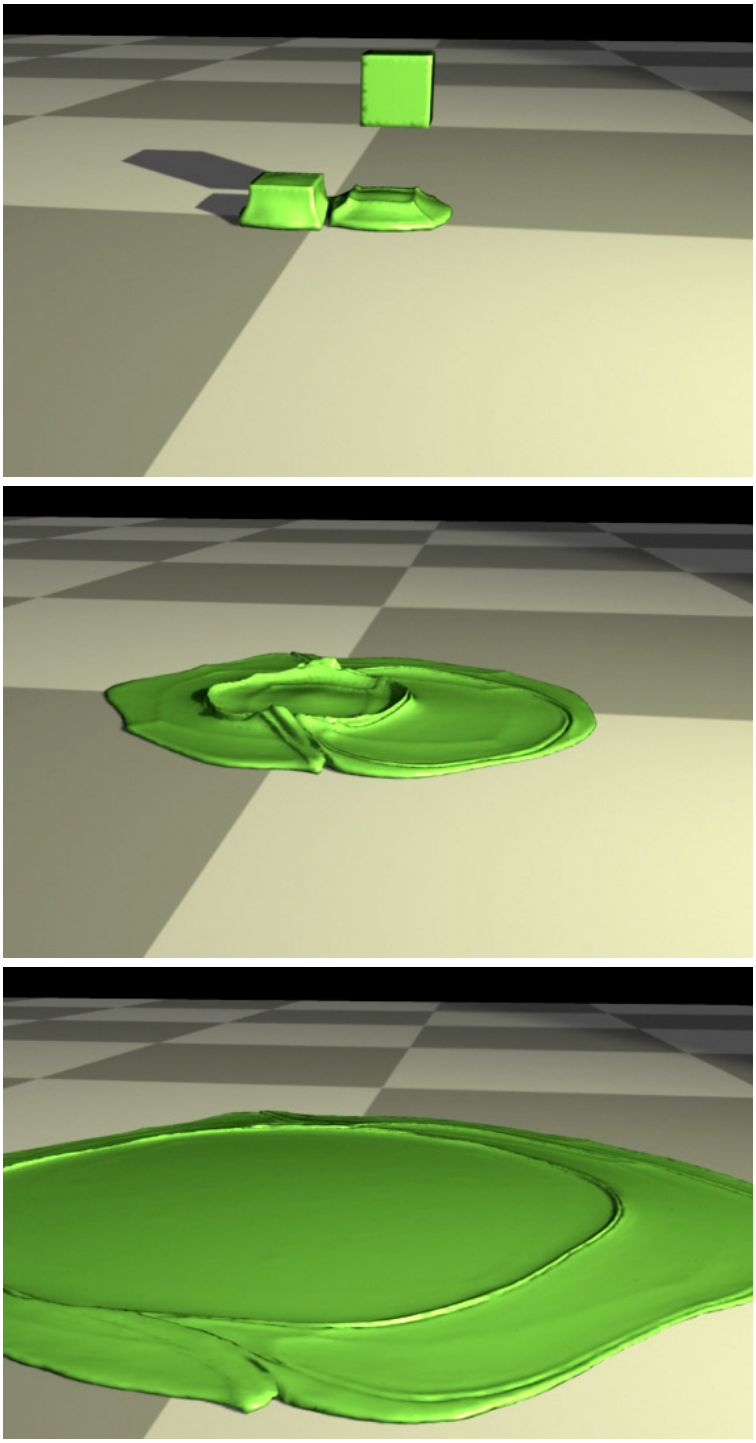


Figure 29: Dropped blocks that merge and spread into a thin sheet.

like handles that lie completely within a cell. Another topological concern is that the extracted isosurface may not match the surface mesh at ambiguous marching cubes faces, leaving a quadrilateral hole in the mesh. We correct this behavior by

triangulating the hole after the mesh surgery, though we could use more principled topological disambiguation in the future. In Section 6.2.3, we proposed a strategy for maintaining thin sheets by re-sampling regions where surfaces merge together, and preserving regions that would split apart. This technique is effective in many of our examples, but it does not properly preserve complicated regions that exhibit both splitting and merging behavior. Specifically, re-sampling these complicated regions can lead to serious popping artifacts, and we offer a solution to this problem in Chapter 7.

As mentioned in Section 6.3.2, the edge-collapses in our stitching strategy can cause robustness problems. In Chapter 7, we will present an alternative mesh-stitching algorithm that avoids edge-collapses entirely, which allows it to completely avoid these gridlock scenarios. Because it is far more robust, we recommend the mesh strategy presented in Section 7.6.1 instead of the method presented in this chapter.

Another drawback to our approach is that the re-meshing depends only on the sampled distance field, and it ignores the actual geometry of the mesh. This means that the joint between the original mesh and the new portions of the mesh can at times appear rough. We propose a smooth subdivision technique to limit these artifacts in Chapter 7, but we think that it may be possible to take into account more information about the original mesh during isosurface creation, and we view this as a fruitful avenue for future work. One final drawback of the approach in this chapter is that the changes to the mesh may result in changes to the volume of the material. For example, when two portions of the taffy are folded together, any empty regions in the cells that are re-meshed will be turned to taffy. This effect is slight if the cell sizes are small, but it can be noticeable for long simulations. There are several approaches to volume control that can correct for this problem, but we have not yet applied any of these methods to our simulator. For example, we could accurately compute the changed volume before and after the re-meshing step and use a method such as [72]

to locally correct this.

We think that our method has a number of strengths that make it attractive for simulating a variety of material types. Level set methods for surface tracking tend to aggressively merge surface components that come close to one another. This has a tendency to wash away the detailed folds that are formed when surfaces are pressed together. Our deep cell test only flags such merge events when they occur deep in the folds, and in most cases it leaves the visible portions of such folds alone. It would be possible to achieve detailed results similar to ours by using a surface tracker that *never* makes any topological changes, but simply allows pushed-together surfaces to lie next to each other. There is a costly consequence to this, however. In such a system, folded materials or complex mixing would result in a huge, overly-complex surface mesh. Most of the geometric detail would be inside folds and would never be seen, and this retained extra detail would be extremely expensive both in terms of memory and computation (updating many more vertices). Our method avoids this pitfall by joining surfaces when they meet and by keeping only the visible portion of the mesh, resulting in a considerable saving in mesh size. Our surface tracker lives between the two extremes of level sets and deforming meshes that do not allow topological changes, and our approach retains the best features of these two approaches.

Many physical simulation methods restrict the amount of detail in the visible surface to that of the underlying physics calculations, and some maintain a separate resolution for the physics calculations and the surface topology calculations. Our method provides three separate detail resolutions: the physics resolution, the topological resolution, and the visible surface resolution. We found this quite useful for exploiting low resolution physics calculations while displaying high resolution topology changes and even higher resolution surface details. We found that the physics simulator was the bottleneck for our computations, because the surface update step is relatively efficient, and because all of our topological calculations are computed

sparsely in time and locally in space. We found it immensely useful to maintain decoupled resolutions for physics, topological changes, and surface detail to effectively provide the illusion of complicated behavior with inexpensive calculations.

CHAPTER VII

PHYSICS-INSPIRED TOPOLOGY CHANGES

In this chapter, we introduce a second method for handling topological changes in triangle meshes that deform through time [155]. The method presented in this chapter builds upon the material covered in Chapter 6 in a manner that produces more physically realistic behaviors for simulations of liquid surfaces. In order to motivate physically correct behavior in our algorithm for computing topological changes, we will first look at how topological changes occur in nature.

7.1 Topology Changes in Two-Phase Flow

The fluid-mechanical term *two-phase flow* refers to a system containing liquid and gas with surface tension forces acting at the interface between them. Liquid and gas are the two different *phases* of matter, and we are primarily concerned with two-phase flow between water (liquid) and air (gas) in this dissertation. The surface tension forces at the interface between water and air often influences the manner in which the liquid surface changes topology.

Most popular methods for tracking a deforming surface ([112], [6], [47], and the material in Chapter 6 [154]), impose topological changes whenever surface features are smaller than a prescribed computational resolution. These resolution-based topological changes cause droplets to pinch off and separate surfaces to merge together. However, because this computational resolution is always several orders of magnitude larger than the width of a water molecule, the resulting topological changes will also cause unnatural phenomena like liquid sheets that rupture and features that completely vanish when they become smaller than a predefined length.

In contrast, liquid surface topological changes in the real world occur when features

become so thin that the continuum model of fluid (described by the Navier-Stokes equations) is no longer applicable. When the distance between individual liquid molecules increases so drastically that their cohesive van der Waals forces forces become insignificant, then the liquid surface will split apart [65]. Likewise, when distant fluid surface components come so close that van der Waals forces forces start to pull their molecules together, then the liquid surfaces merge.

Although the actual change in topology is caused by nanoscale forces representing a breakdown of the continuum model, these topological changes will not take place unless the surface geometry is already microscopically thin. Therefore, in order for topological changes to take place, some large-scale physical forces must lead to the formation of geometric surface structures that are only a few molecules wide. Such thin structures invariably result from natural *instabilities* in the governing equations for fluid flow.

7.2 Natural Instabilities in Two-Phase Flow

Some forces in the Navier-Stokes equations directly compete with one another. For example, surface tension forces aim to minimize surface area, while pressure forces aim to preserve volume. When these forces perfectly balance anywhere in the system, that region is said to be in *equilibrium.* These equilibria can be either *stable* or *unstable.*

When it takes a significant increase in energy to leave an equilibrium state, the system is said to be *stable*; if a stable system is slightly perturbed, the fluid forces will act to restore the original state of balance. An example of this type of stable equilibrium is a motionless glass of water sitting on a table. If the glass is jostled, or if the liquid surface is upset by a light breeze, the liquid in the glass will soon return to its original stable state.

When a slight perturbation can cause an equilibrated system to produce large forces and completely abandon its current state, the original equilibrium is said to be

unstable. Several instabilities can occur in two-phase flow, and some of them produce forces that shrink geometric surface features into an infinitely thin singularity. Of course, infinitely thin liquid structures do not exist in nature — at a small enough scale, the continuum model breaks down and these thin structures simply rip apart, causing a change in topology.

To make this idea more concrete, we will first discuss the nature of the most relevant natural instabilities, and then we will discuss the stability of each type of small surface structure individually.

7.2.1 Rayleigh-Plateau Instability

The first instability that we will discuss in this chapter is the Rayleigh-Plateau instability. This instability occurs when surface tension forces begin to dominate the fluid's inertial forces.

Imagine a long, thin cylinder of liquid suspended in space. The pressure forces act to preserve volume, while the surface tension forces act to minimize surface area. In the case of this cylindrical structure, the surface curvature is constant everywhere (because the cylinder's radius is constant along the long axis), and the volume is evenly distributed. The surface tension and pressure forces balance, so the system is in equilibrium.

Now imagine that we perturb the cylinder slightly by decreasing its radius in one spot. The radius of curvature at this location is much higher than other nearby locations, so the surface tension force is much higher at this location as well. The pressure force now seeks to maintain a constant liquid volume with as little effort as possible. Trying to maintain a constant radius everywhere requires a great deal of energy, because it has to fight this large surface tension force. Instead, the liquid can flow away from this high-curvature region and bulge out the nearby regions which have lower surface tension forces. This requires less energy overall, and it is what

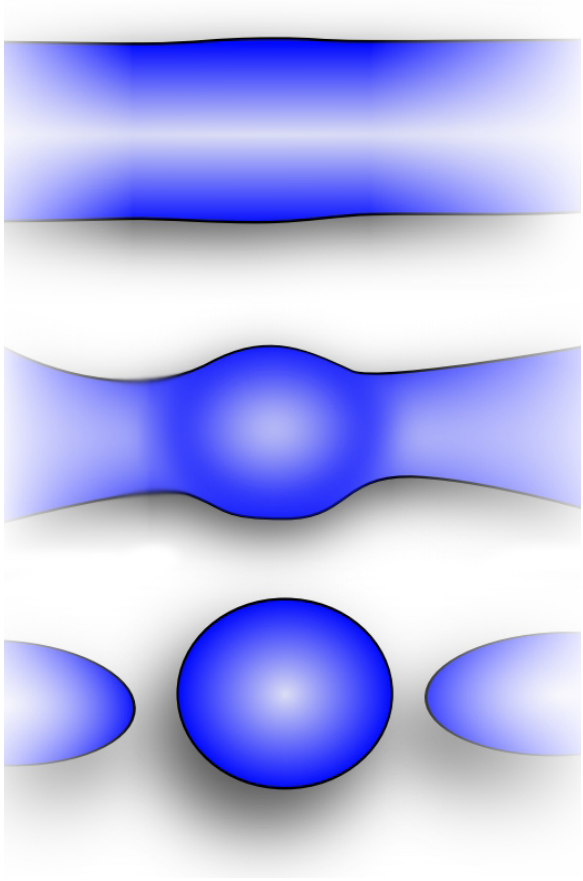


Figure 30: Illustration of a Rayleigh-Plateau instability developing from top to bottom. At the start (top), small variations occur in a cylindrical tube. Next (middle), surface tension forces exaggerate the small perturbations, until (bottom) the tube eventually pinches off into droplets.

happens in nature.

When small perturbations are introduced into thin cylindrical liquid structures, liquid flows away from the disturbance and the perturbation amplifies. The imbalance in forces amplifies as well, creating even larger surface tension forces. This behavior continues until the radius becomes infinitely thin and has nowhere left to go. In nature, the continuum breaks down some point before the cylinder becomes infinitely thin, and individual molecules spread apart, causing a topological change. Figure 30 illustrates the topological change caused by a Rayleigh-Plateau instability.

The Rayleigh-Plateau instability can be seen easily at home by setting your kitchen

sink to slowly drip. Each one of these droplets results from a Rayleigh-Plateau-induced topology change. In fact, almost every water droplet that you see in your life will be caused by a Rayleigh-Plateau instability. When a wave crashes against rocks on the beach, the water thins out into a thin sheet, and the boundary of that thin sheet rounds out into a half-cylinder. Any variations in this boundary shape will grow and quickly pinch off water droplets. The iconic "crown" shape that we are used to seeing at the ridges of a splash is also caused by the Rayleigh-Plateau instability [38].

This *Rayleigh-Plateau instability* can be predicted by a dimensionless parameter called the liquid's *Weber number*:

$$We = \frac{\rho v^2 l}{\sigma}, \tag{14}$$

where ρ is the density of the liquid, v is the liquid's velocity, l is a characteristic length of the liquid (the diameter of a cylinder or thickness of a liquid sheet), and σ is the surface tension coefficient. These values can each vary at different points throughout space, so the Weber number can change depending on where you measure it. The Weber number characterizes the ratio of the liquid's kinetic energy to surface tension energy. Consequently, when the Weber number drops below $We = 1$, the surface tension forces dominate and the water breaks into droplets due to Rayleigh-Plateau instabilities [32, 36, 37].

7.2.2 Convective Instability

The second type of natural instability in this chapter is the *convective instability* [85, 84]. While the Rayleigh Plateau instability acts at small scales, when the Weber number drops below 1, the convective instability acts on large structures with $We > 1$.

Imagine you are creating an arc of water by aiming a high-powered hose away from your body. If you hold the source of this flow (the hose) still, then the flow will reach a steady state. On the other hand, any perturbation to the source of the

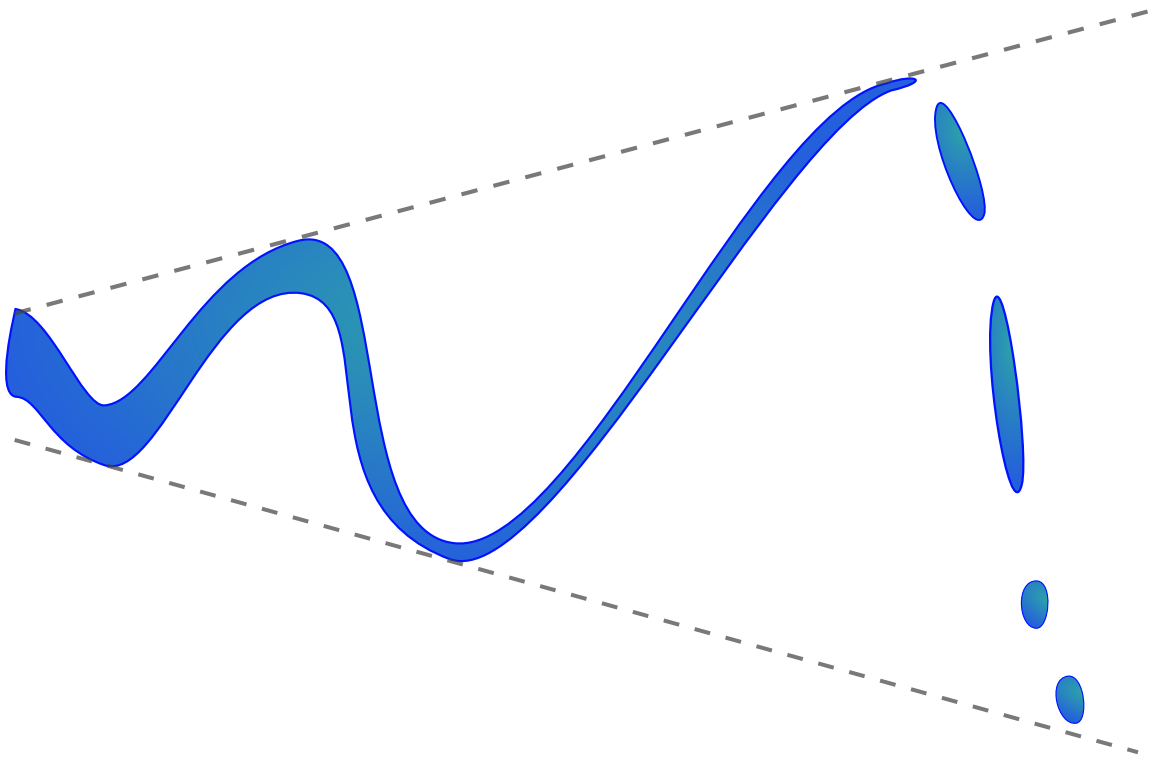


Figure 31: An illustration of a convective instability. Small perturbations in the initial stream (left) are naturally exaggerated as the fluid convects from left to right. Because a constant volume of liquid is stretched apart as it travels from left to right, the stream becomes thinner. Eventually the fluid stream can become so thin that its Weber number drops below unity and Rayleigh-Plateau instabilities can develop into an absolute instability (far right).

flow will propagate through the resulting stream of water. In fact, small perturbations introduced in one region of the flow will actually amplify as they are convected downstream. If you wiggle the angle of the hose slightly, the water stream spreads out further and further the longer it travels through the air. This effect is visualized in Figure 31.

As the amplifications to the stream get larger and larger, the liquid spreads thinner and thinner. These thin bodies of liquid gain a much larger surface area-to-volume ratio, and their Weber number drops dramatically. Eventually, Rayleigh-Plateau instabilities take over and rip the stream of liquid into droplets. Although convective instabilities are not directly responsible for topological changes in liquid, they are one method for large, heavy bodies of liquid to naturally thin out into smaller structures with smaller Weber numbers.

7.2.3 Rayleigh-Taylor Instability

The final type of instability that I will discuss in this chapter is the *Rayleigh-Taylor instability* [120, 140]. These instabilities apply in a system with two or more immiscible fluids of different densities (like oil and water, or like water and air). We will only consider two phases (air and water) in this chapter, so we will limit the rest of this discussion to the case of only two-phase flow. The denser of the two fluids has more mass per unit volume, so the force it experiences due to gravity (or any other constant acceleration) will be proportionately larger than the lighter fluid. To see this, we can rearrange Newton's second law $f = ma$ to see that the force per unit volume is equal to the density times acceleration, $\frac{f}{V} = \rho a$. We perceive the lighter phase of fluid to be more "buoyant" than the denser one because of these unequal forces.

Pressure forces will compete with this gravitational force in order to ensure that the fluids maintain their volume. Although each fluid phase will experience forces in

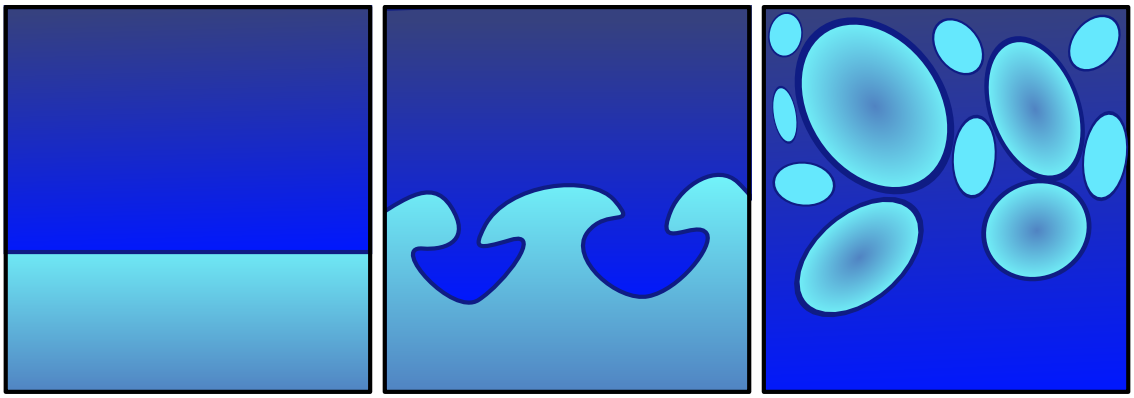


Figure 32: A Rayleigh-Taylor instability develops when two fluid phases of different densities experience forces that cause them to trade places. This illustration shows how water (dark blue) can trade places with air (light blue) in such an instability. Initially (left), the water is perfectly balanced on top of a region of air. Small imperfections become larger (middle) as the fluids find a way to trade places. Finally (right), the air phase ends up on top and the water phase ends up on the bottom. Surface tension at the interface between the two flows can cause air bubbles and water droplets to form as the different phases pass through each other.

the direction of the external acceleration, the pressure force prevents these fluids from occupying the same space. The gravity and pressure forces can achieve equilibrium when they perfectly cancel each other out.

A stable equilibrium is achieved when the densest fluid settles to the bottom and the lighter fluid balances on top. Any perturbation to this state will result in the system eventually returning to this stable configuration.

An unstable equilibrium occurs when the denser fluid phase balances perfectly on the lighter fluid phase. Although the forces perfectly balance out in this scenario, it is a highly unstable equilibrium. Any small perturbation at the interface between the two phases will result in the denser fluid trading places with the lighter one, as seen in Figure 32.

Although we have used gravity as the external acceleration in the above discussion, Taylor showed that any acceleration will behave similarly [140]. For example, if we allow a closed container full of oil and water to come to equilibrium, the water will

settle to the bottom. If we then accelerate the container downward faster than gravity, the container will feel gravity-like acceleration upward. A Rayleigh-Taylor instability would then be induced, and the two fluids would trade places. Halting the acceleration would then cause another instability and another mixing of fluids.

High pressure forces can also induce Rayleigh-Taylor instabilities. For example, imagine two walls of water colliding into each other from the left and right. The two phases of dense water trap a phase of lighter air in between them as they collide. The pressure and momentum of the water induce a Rayleigh-Taylor instability as the liquid tries to trade places with the air.

Rayleigh-Taylor instabilities can create extremely detailed tiny structures when the two fluids accelerate past each other. In particular, many finger-like tendrils of fluid can result. If a strong enough surface tension exists at the interface between the denser and lighter fluids, Rayleigh-Plateau instabilities can induce topological changes (Section 7.2.1). In the case of two-phase flow like water and air, these topological changes form water droplets and air bubbles. Thus, Rayleigh-Taylor instabilities are another natural instability that can lead to topological changes in two-phase flow.

7.2.4 Stability of Small Geometric Structures in Two-Phase Flow

Now that we have discussed several natural instabilities that can lead to topological changes in a two-phase flow, we will investigate which geometric structures are vulnerable to these topological changes, and which geometric structures are stable. First of all, we know that any large structures that eventually change topology must first transition to smaller or thinner structures before changing topology — only small or thin structures will lead to topological changes. As a result, we only need analyze the stability of very small geometric structures that can arise in two-phase flow.

We will perform our analysis by first sorting all possible shapes into four broad classes: volume-like shapes, sheet-like shapes, thread-like shapes, and point-like shapes,

as illustrated in Figure 33. Each of these shape classes represents a number of dimensions in which the shape is extremely thin: volume-like shapes are very large and thick, while sheet-like shapes are vast in the two dimensions coplanar with the sheet but thin in the dimension normal to the sheet. Likewise, thread-like shapes are very long in one dimension but thin in the other two, and point-like shapes are very small in all three dimensions.

The geometry of these shapes plays a large role in determining its stability, but the relative density of the shape matters as well; denser fluid phases tend to dominate in Rayleigh-Taylor instabilities. As a result, we will analyze the stability of each of these shape classes formed both by liquid surrounded by gas as well as gas surrounded by liquid, for a total of eight cases. We will assume that the density of the gas is negligible compared to that of the liquid (water is about one thousand times as heavy as air — $1000kg/m^3$ compared to $1.2kg/m^3$).

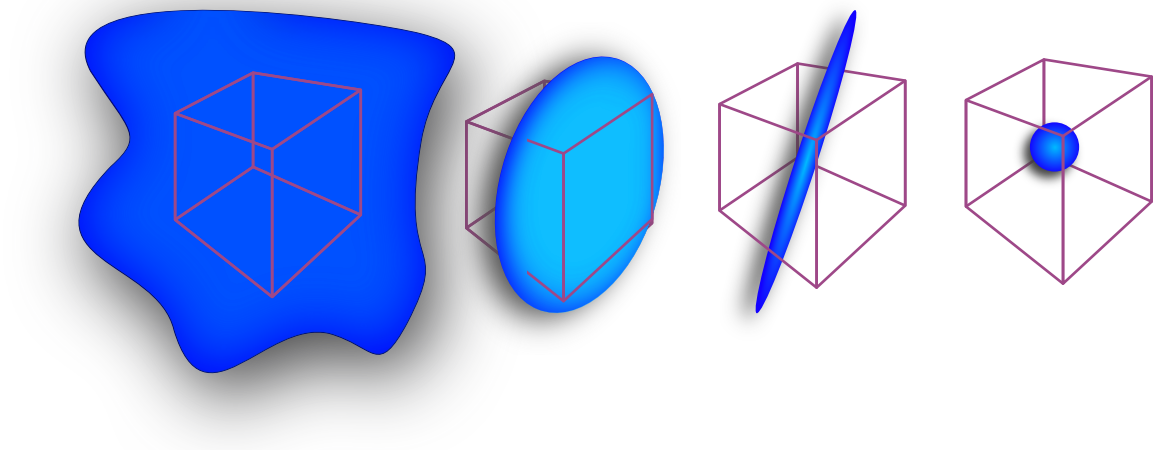


Figure 33: Classes of geometric structures (from left to right): Volume-like fluid, sheet-like fluid, thread-like fluid, and point-like fluid. The blue objects represents the surface of a geometric structure, with the purple cube encapsulating the region of interest.

Volume-like liquid: The first case we will analyze is a very large body of liquid.

Because we are not considering any small structures here, surface tension effects are negligible. Even if we introduce a small perturbation to the smooth boundary of a body of water, surface tension (which aims to minimize surface area) will remove the perturbation and return the liquid to its original state. Topological changes will not occur unless more mixing between the different fluid phases creates small geometric structures, so this case is topologically *stable*.

Volume-like gas: Large bodies of gas are topologically *stable* for the same reason as liquids — more mixing needs to occur before structures are small enough to lead to topological changes. If we slightly perturb the edge of a large bubble of air, surface tension will return the interface to its original state.

Sheet-like liquid: Here we consider a thin sheet of liquid (water) with gas (air) on each side of the sheet. If we slightly perturb the surface of this sheet, surface tension will return the sheet to its original state, so Rayleigh-Plateau instabilities do not apply to this geometry (Rayleigh-Plateau effects are very significant along the boundary of this thin sheet, where the curvature is high, but they do not apply to the large flat region at the center of this sheet). The momentum of this denser liquid phase tends to be less vulnerable to Rayleigh-Taylor instabilities as well.

Convective instabilities can cause these liquid sheets to become thinner and thinner over time, at which point the edges of this geometry have already eroded and formed smaller droplets due to Rayleigh-Plateau instabilities. An "absolute instability" [85, 84] will quickly occur once the edges of the sheet begin ripping apart into droplets, causing the entire sheet to disintegrate. We do not need to consider this instability in this case, because the thread-like liquid (below) will already account for the transition of the sheet boundary into droplets. In this case, we only need to consider the extremely rare case where a convective instability thins the sheet out so much that it ruptures due to a lack of water molecules.

Overall, the thin liquid sheet geometry is remarkably *stable*. To give an intuitive

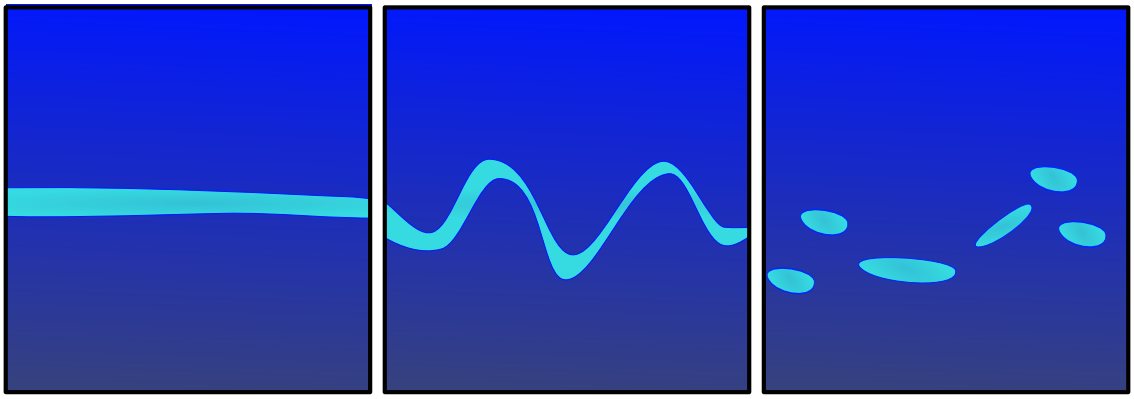


Figure 34: A Rayleigh-Taylor instability can affect a thin sheet of air trapped between two larger bodies of water. Small perturbations get exaggerated (left, middle) until the sheet breaks up due to surface tension forces (right).

example, consider the thin film of liquid surrounding a soap bubble; it highly resistant to small perturbations and only ruptures when the film becomes drastically thinner than the thickness required for an absolute instability.

Sheet-like gas: Contrary to the stability of the sheet-like geometry of liquid, a thin sheet of gas surrounded by much heavier liquid is significantly less stable. The main reason for this discrepancy is the Rayleigh-Taylor instability. When slightly perturbed at the boundary of a thin sheet of air, surface tension will indeed counteract the perturbation. However, the huge amount of additional momentum imparted by the denser liquid can dominate the surface tension forces and rip the gas sheet apart.

Imagine a thin sheet of air perfectly balanced between two large bodies of water in a sealed container. The denser water on top will force itself through the sheet of air under the will of gravity, creating many smaller bubbles in the process (See Figure 34). Because a Rayleigh-Taylor instability is significantly more likely to develop in a thin sheet of air than in a thin sheet of liquid, we will consider this sheet-like geometry for a gas phase relatively *unstable*.

Thread-like liquid: Now we consider a very thin cylinder of liquid surrounded by air, as in Section 7.2.3. As explained above, this geometry is highly *unstable* due

to a Rayleigh-Plateau instability, and it will rip apart into droplets of water.

Thread-like gas: If we trade the places of the fluid phases in the previous example and create a thin cylinder of air surrounded by water on all sides, a Rayleigh-Plateau instability still develops. The thread of air rapidly decomposes into a series of air bubbles. This geometry is *unstable*.

Point-like liquid: Next, we consider a small sphere-like geometry of liquid surrounded by air. This geometry is commonly seen throughout nature as a tiny droplet of water. At small scales, the continuum forces are completely dominated by surface tension. Because the surface area is minimal in the shape of a sphere, any small perturbation will quickly return to its original spherical shape due to the surface tension forces. This geometry is *stable*.

Point-like gas: Finally, we consider a small sphere-like structure of air surrounded by water on all sides. This shape is commonly seen in nature as a bubble. Just like the point-like structure of liquid, this small geometry is dominated by surface tension forces, and it is extremely *stable*.

7.3 Application to Fluid Simulation

We would like to apply the geometric analysis in the previous section in order to better handle topological changes to the fluid simulation method in Chapter 4. Essentially, we would like to improve upon the idea from Chapter 6 of locally analyzing small cubic samples of space and replacing unacceptable geometric structures with surfaces of simpler topology. However, instead of comparing the surface to a low-resolution implicit surface, we will create new rules that preserves stable geometric structures like thin sheets of liquid, and we will consequently have to use more sophisticated surface reconstruction procedures than a simple "marching-cubes"-style lookup table.

Our work begins with a method for explicitly tracking a fluid surface, similar to the methods of Du et al. [42] and Wojtan et al. [154] (Chapter 6). These techniques

deform a high-resolution surface mesh and locally detect and correct topological inconsistencies by re-sampling from a grid-based signed distance function. This tactic successfully preserves high resolution surface details in topologically simple regions of the surface, but it relies on marching cubes [88] to re-sample from a signed-distance function at sites where topology changes occur. Unfortunately, because marching cubes cannot represent features smaller than a grid cell, this approach cannot perform topological changes on thin features without deleting large sections of the surface at a time.

The technique presented by Müller [98], on the other hand, is capable of performing topological changes on some types of thin features, because it uses an extended set of marching cubes tables that is specifically designed to permit sheets thinner than the grid resolution. However, this strategy still cannot represent certain small features like thin columns or small droplets of liquid that do not intersect the edges of the marching cubes cells. Furthermore, this method re-samples the surface for all cells in every time step so it will not preserve small-scale surface details through time.

We propose a mesh-based surface tracking method for fluid animation that both preserves fine surface details and robustly adjusts the topology of the surface in the presence of arbitrarily thin features like sheets and strands. Instead of using lookup-table-based re-sampling methods to generate a surface from a topologically valid signed distance function, we introduce a local convex hull method for connecting surface features during topological changes. We sew this new surface together with the original surface mesh using a subdivision-based mesh-stitching algorithm, and then we create new vertices by subdividing these newly-created triangles until the surface is adequately sampled. The contributions of this chapter are as follows:

- **Local convex-hull algorithm for topological operations** We combine information from both the input mesh and a volumetric signed distance function in order to reconstruct a physically-valid topology that is easily controllable.

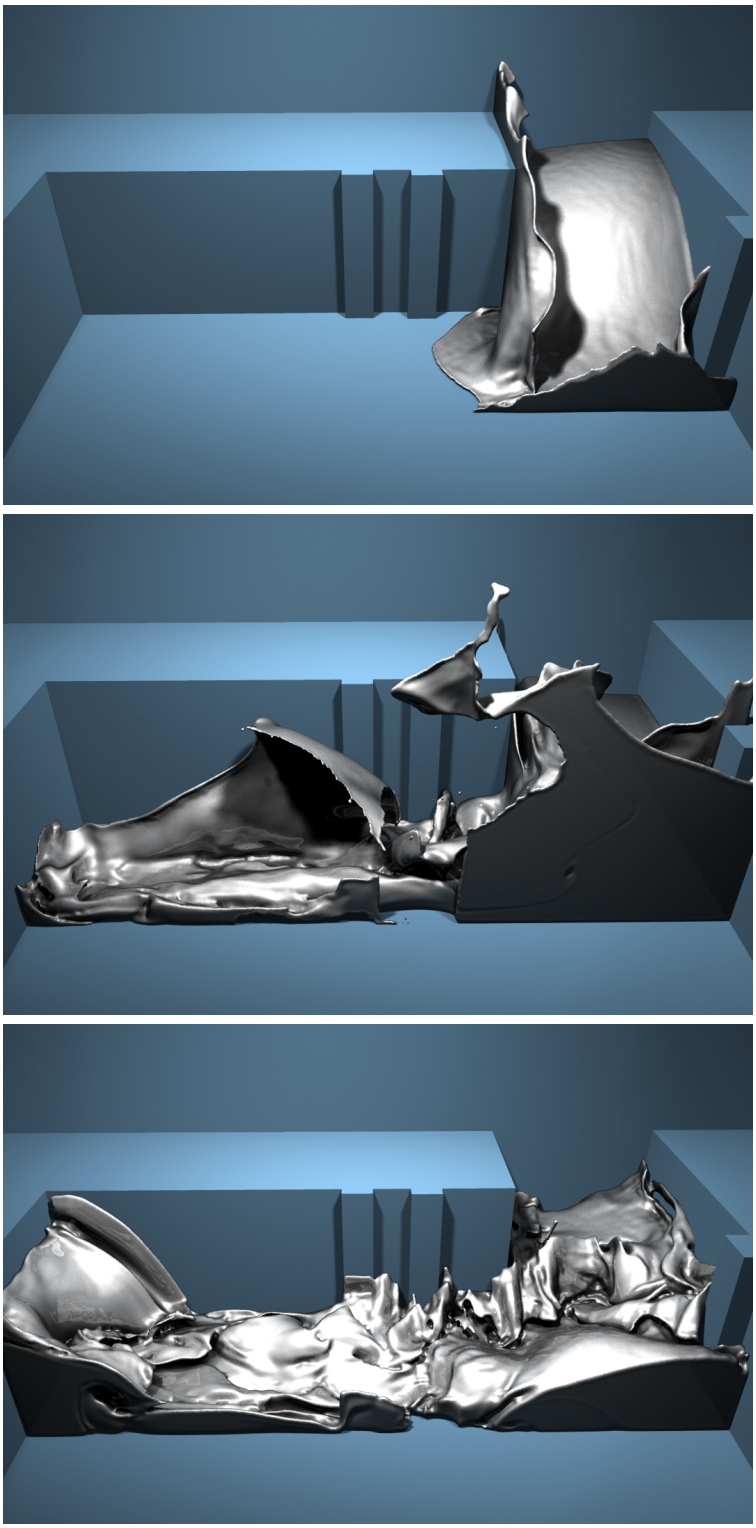


Figure 35: Our algorithm efficiently produces detailed thin sheets and liquid droplets, even with low-resolution fluid simulations — The main corridor in this example is only 30 fluid cells wide.

- **Detail preservation** We avoid re-sampling the surface everywhere except where topological changes occur. In those locations, both our local convex hull operation and a subdivision-based method for stitching together surfaces preserve as many Lagrangian surface details as possible. In places where we cannot avoid re-sampling the mesh, we apply an interpolating subdivision scheme to ensure a smooth surface surface.

- **Thin Features** Our method naturally produces arbitrarily thin fluid features like sheets and strands while preserving high-resolution surface details.

- **Constrained Topology** Our algorithm outputs a high resolution surface mesh whose topology is constrained to match that of a lower resolution fluid grid. This construction allows for the efficient simulation of highly detailed surface animations while preventing many potential artifacts caused by inconsistent mesh and grid topologies.

7.4 Fluid Simulation with an Explicit Mesh

To review the content in Chapter 4, we use an Eulerian fluid simulation to advect a Lagrangian triangle mesh, and we perform local mesh operations like subdivision and edge collapses in order to ensure reasonably high quality triangle shapes. To inform the fluid simulation which cells should be simulated as liquid, we use the surface mesh to create a signed distance function with samples collocated with the pressure values in the fluid grid.

We employ a voxelization-style method [98] to compute the signed distance function. We first voxelize the triangle mesh onto a grid by computing intersections with a rays in the x, y, and z directions. For each ray, we keep track of its inside/outside status with a counter. At the start of the ray (which is guaranteed to be outside of the mesh), the counter has a value of zero. For each intersection with the triangle mesh, we increment the counter if it intersects a triangle whose normal is facing the

opposite direction of the ray, and we decrement it if it intersects a triangle with a normal facing the same direction of the ray. This way, all regions outside of the mesh will have a counter value of zero, regions inside the mesh will have a value of one, and inside-out regions will have values less than zero or greater than one. We store this counter value at each grid point to assign an inside/outside status, and then we compute the exact distance from the point to the surface in order to complete the signed distance calculation.

Every pressure value in the fluid grid that is collocated with an "inside" value is to be treated as an active fluid cell. Very thin features may not have any "inside" values nearby, so we additionally mark any cells that overlap the surface as active fluid cells. We then compute new velocities with the fluid simulation and use them to

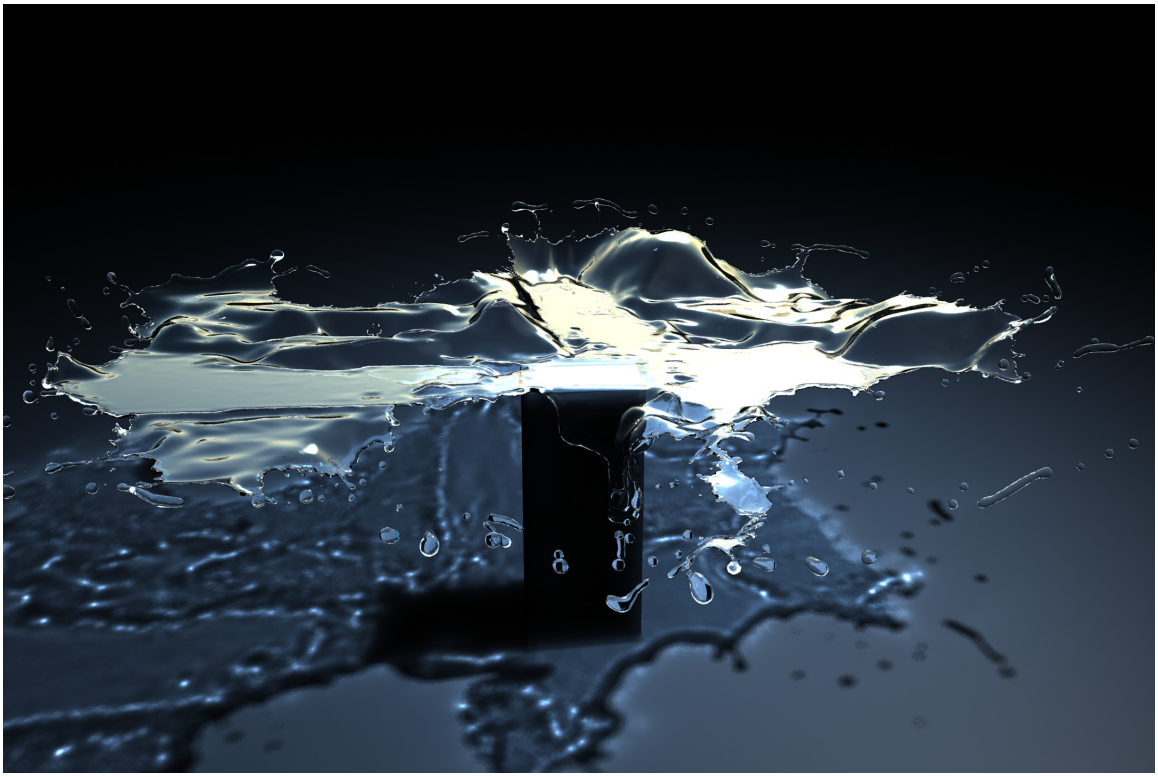


Figure 36: When combined with a mesh-based surface tension technique, our method produces realistic breakup behavior of thin liquid films. In this example, a ball of water smashes downward into a rectangular podium, rapidly spreads out into a chaotic sheet, and breaks up into droplets.

advect the surface mesh using an explicit Runge-Kutta technique. Finally, at the end of each time step, we detect and correct the topology of the surface. The remainder of this chapter will explain our novel method for handling topological changes.

7.5 Topological Connectivity

The main goal of changing the topology of the mesh is to ensure that the liquid surface is connected to other regions of fluid in the same way as the pressure values. If these topologies differ, then distracting visual artifacts will occur. For example, if two disconnected surface components lie within the same cell in the fluid grid, then the fluid pressure values will be unable to distinguish between these independent components. The low resolution fluid velocities will then move both surface pieces together in the same direction, creating an invisible link between them that tends to persist for the entirety of the simulation.

7.5.1 Defining Valid Topology

We detect disagreements between the explicit surface mesh and the fluid grid by locally contrasting the topology of the explicit surface with a surface that possesses the same topology as the fluid simulation. We define any region where these topologies disagree as *topologically invalid* (note we are creating a direct analogy to the term "topologically complex" from the previous chapter). As mentioned in Section 7.4, the fluid volume is defined by the inside/outside values of a signed distance function in addition to all cells that overlap the surface geometry. If the surface geometry contained no thin features, then we could simply compare the topology of the explicit surface to the topology of the isosurface of its signed distance function, as in Chapter 6 [154]. However, we wish to preserve thin features that cannot be resolved by the signed distance function, so we must use a more versatile definition of topological validity.

Before specifying what it means for surface geometry to be topologically valid,

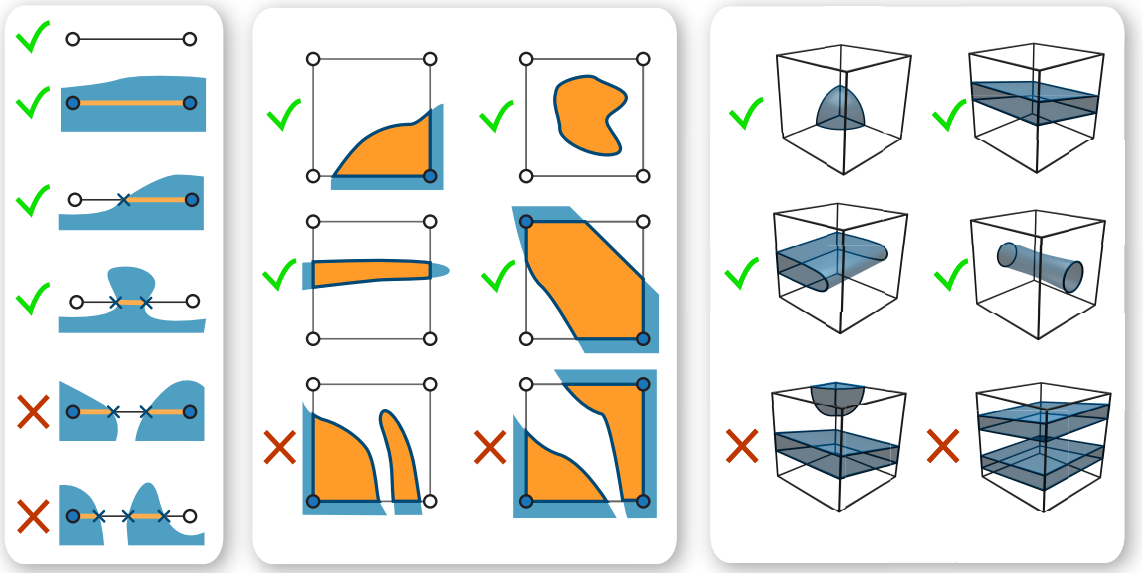


Figure 37: Examples of valid (green check mark) and invalid (red X) topology for edges, faces, and cells.

we first define a topological cell as a cube with its eight corners located at sample points in the signed distance function. The cube is bounded by six faces, twelve edges, and eight corners. We analyze the topology of the explicit surface mesh by examining the intersection between the the solid geometries of the triangle mesh and each topological cell. This intersection is empty if the cell is completely outside of the surface, the intersection is identical to the original cube if the cell is completely inside the surface, and the intersection is more complicated if the cell overlaps the surface.

In order for the cell to be topologically valid, the surface of this intersection should have the same connectivity with its neighbors as the fluid cell does, and it must not have any more topological features than a single fluid cell. That is, it must contain at most one surface component, with no holes or voids — the surface of this intersection should be homeomorphic to a sphere (more formally, a 2-sphere). Similarly, the transitions from this topological cell to its neighbors must be topologically valid, so the surface intersections with faces and edges of this cell should be homeomorphic to

1-spheres (circles) and 0-spheres (intervals), respectively. Lastly, a corner of the cell is topologically valid if it is not located in an inside-out region of the surface. Figure 37 shows some examples of valid and invalid geometry.

We can efficiently detect the topological validity of a cell corner by checking its counter from the signed distance function calculation in Section 7.4 The counter of a valid corner must be either zero or one, otherwise it means the surface is inside out or self-intersecting. Next, we determine the validity of a cell edge by counting the number and orientation of its intersections with triangles in the surface mesh and comparing the result with a single line segment with outward-oriented endpoints (0-sphere). If there are too many components, or if the surface is oriented the wrong way at any of the intersection points, then the edge is invalid. This step is performed at the same time as the corner validity test (during the creation of the signed distance field). We can check the validity of a face by computing its intersection with the mesh and counting the number of components, and we can test the topological status of a cell by similarly counting connected components and ensuring that the Euler characteristic detects no holes.

The corner validity test samples the signed distance function at several regularly-spaced sample points, and it is guaranteed to identify any self-intersections larger than the grid spacing. This test will catch any topological flaws that are well-resolved in the x, y, and z dimensions, just like marching cubes will faithfully reproduce a surface as long as there are no features smaller than the grid size. The edge validity test checks all of the edges in the grid, so it is guaranteed to identify any topological flaws that are well-resolved in at least two dimensions. This means that self-intersections and pockets of air that look like thin sheets will be identified by the edge validity test. This test can also catch thin spindles and voids if they happen to intersect one of the grid edges. The face validity test checks for intersections with all faces in the grid, so it is guaranteed to catch topological flaws that look like thin spindles (which

span more than one cell in a single dimension, but are very thin in the other two). However, the face validity test cannot catch topological flaws smaller than a single grid cell unless they happen to intersect the face. Finally, the cell validity test is guaranteed to identify all topological flaws, because it checks within every cell.

7.5.2 Topological Detection In Practice

In practice, such topological problems smaller than a grid cell do not exist, because we start with a well-resolved surface and then smoothly deform it according to a low resolution fluid velocity field. This means that large features morph into small features through a gradual process, by first becoming thin sheets and then thin spindles. Because topological inconsistencies are caught by lower-dimensional validity tests before they have time to shrink smaller than a grid cell, we have not found it necessary to perform the full cell validity test.

Face validity tests are mostly redundant for our purposes as well, because perturbations from the fluid velocity tend to force thin structures to intersect cell edges — the edge and corner validity tests tend to quickly catch all problems. As a result, we have found it practical to bypass the testing of any cells and faces unless we specifically have to guarantee valid topology in a particular region, as we will describe shortly.

To summarize the work done in practice by our implementation, we only perform the topological validity tests on the corners and edges for every corner and edge in the fluid grid. That is, we check all corners to see whether they represent inside-out geometry, and we check all edges to see if the surface of their solid intersection is homeomorphic to a 0-sphere. To optimize our implementation at the expense of missing some small geometry within a single cell, we do *not* perform topological validity tests *within any cells*. The test to guarantee that the surface of the solid geometry is homeomorphic to a 2-sphere requires computing the boolean intersection

between a cell and the mesh, counting the number of holes, and then counting the number of components. Although this test would help guarantee that all of our geometry is simple enough to be represented by a fluid cell, it does not catastrophically break our algorithm if we neglect it either. As a further optimization, we do *not* test every *face* for topological validity either. Because this test is not trivial (it requires intersecting the mesh with a face and counting the number of connected components in this intersection), we only perform it at the boundary between topologically valid and topologically invalid cells, as explained in the next paragraph.

After an invalid cell is detected, we will soon replace its intersecting surface with a similar surface that has topologically valid geometry. Because this cell shares its boundaries with other cells that may not have been classified as topologically invalid, we have to specifically guarantee the validity of its faces. In this case, we count the face components and pass topological validity information to neighboring cells, similar to the complex cell propagation strategy in Chapter 6 [154]. To summarize the frequency of topological tests in our implementation: we exhaustively test every corner and edge in the signed distance grid, we never check cells, and we only check faces when it is absolutely necessary to ensure valid connectivity between an invalid cell and its topologically valid neighbors.

7.5.3 Correcting Invalid Topology

After deciding that a cell has invalid topology, we must replace the surface/cube intersection in that cell with a new one. We require that this new intersection surface meets two constraints: it should be topologically valid, and it should cleanly connect with the original surface. In addition, because each vertex of the original surface represents a valuable piece of Lagrangian simulation data, we also desire that the new surface preserves as many of the original surface vertices as possible.

According to our definition of topological validity, a valid intersection must have

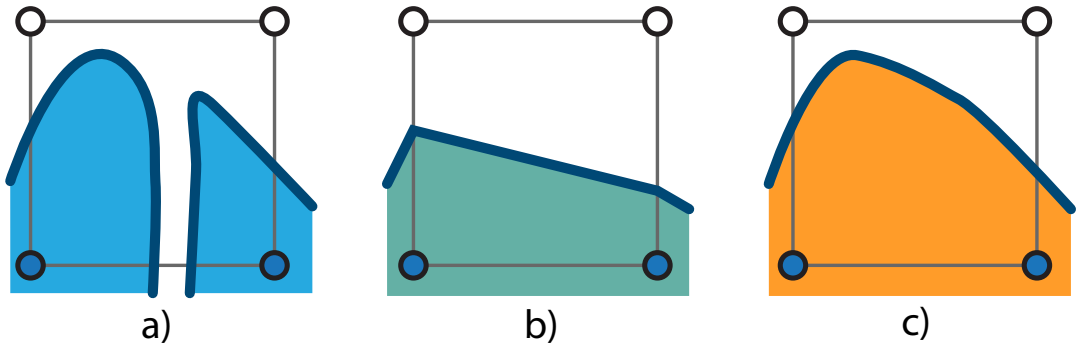


Figure 38: Given a surface with invalid topology (a), methods based on marching cubes-style lookup tables will ignore interior vertices (b), while our method preserves many of the original details (c).

a surface which is either either empty or homeomorphic to a 2-sphere, its intersection with each cell face must either be empty or have a boundary that is homeomorphic to a 1-sphere, and it intersection with each cell edge must be either empty or have a boundary which is homeomorphic to a 0-sphere. A straightforward strategy for reducing the geometric complexity of a surface/cube intersection is to wrap a single surface around all of the original surface components. Fortunately, this new surface can be created efficiently by replacing the original surface/cube intersection with its convex hull. If the input surface intersected the cell boundary, then the convex hull preserves this connectivity. Furthermore, the convex hull's intersections with faces and edges are lower dimensional convex hulls, so they are also topologically valid. In addition, all of the vertices on this convex hull are preserved from the input surface, so we reduce re-sampling errors during this process, as illustrated in Figure 38.

In practice, we use the qhull library [5] to compute the convex hull of the following points: all original surface vertices that lie within that cell, the vertices created by intersecting the original mesh with the edges and faces of the cell (described in Section 7.6), and all "inside" corners of the cell. Finally, because the convex hull represents the intersection between a cube and the new surface, we extract the final surface by

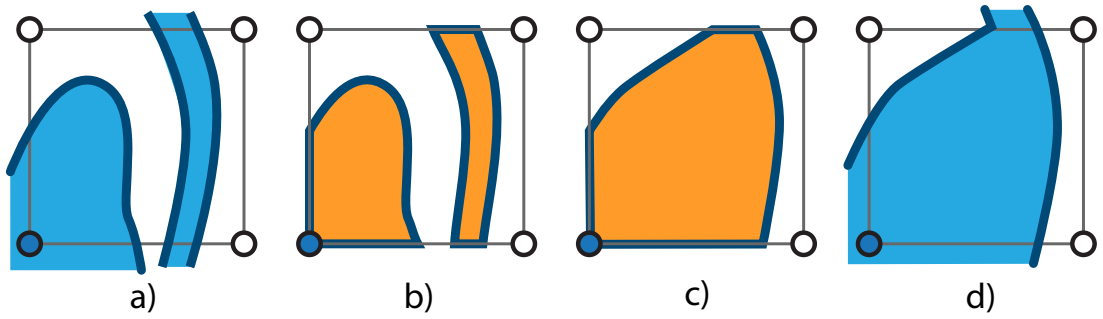


Figure 39: Given a triangle mesh and topological cell (a), we examine the intersection between the the solid geometries of the mesh and the cell (b). If the intersection has invalid topology, we replace it with its convex hull (c). Finally, we remove facets belonging to the boundary of the cell and reconnect the surface (d). Note that the cell below this one will also be re-sampled, because the bottom edge in (a) is topologically invalid.

deleting any facets that are co-planar with the cell boundary. See Figure 39 for an example.

This strategy of reducing invalid surface regions to topological spheres has physical implications when used in a fluid simulation: (1) thin sheets of air will be detected as topologically invalid and destroyed, and (2) thin sheets and strands of liquid will persist throughout the simulation. We will discuss how to further control the final surface topology in Section 7.5.5, and we will analyze the physical plausibility of all resulting shapes in Section 7.5.5.1.

7.5.4 Morphological Interpretation

The algorithm presented in this chapter can be expressed in terms of mathematical morphology [128]. We can think of our background grid G as a complex of cells of dimension $d = 0, 1, 2, 3$. The 3-cells are the cubic cells themselves, the 2-cells are the faces of these cubes, the 1-cells are the edges, and the 0-cells are the vertices where the grid lines intersect. Ignoring the practical optimizations presented in Section 7.5.2, we formally describe the morphological steps that our algorithm will take in Figure 40.

```
1   Given a grid G, which is a complex of d-cells
2
3   For each 3-cell C in G
4       L(C) := false
5
6   For d = 0 to 2
7       For each d-cell C in G
8           if (C ∩ S !≈ d-◯)
9               For each 3-cell Q adjacent to C
10                  L(Q) := true
11
12  For each 3-cell C in G
13      if (L(C) ∨ (C ∩ S !≈ 3-◯))
14          S := S∪H(C ∩ S)
```

Figure 40: Ideal algorithm for topological changes in terms of morphological operations

First, in lines 3–4, the algorithm runs through all of the cubes in the grid and ensures that they will not be re-sampled by default. It does this by setting each cube's label $\mathrm{L}(C)$ to `false`.

Next, starting at line 6, the morphological algorithm runs through all vertices, edges, and faces in the grid (0, 1, and 2-cells) and computes the solid intersection with the volume S enclosed by the surface mesh. The algorithm then checks whether there exists a homeomorphism between this intersection and a ball of the same dimension (We use the symbol $\approx$ to denote a homeomorphism, and we use $!\approx$ when no homeomorphism exists). This means we check each 2-cell (cube face) to see if it is homeomorphic to a disk, and we check each 1-cell (cube edge) to see if it is homeomorphic to a line segment. In Figure 40, we represent a ball of dimension d with the notation d-◯. Note that the surface of a d-ball is a $(d-1)$-sphere, and the notion of a solid being homeomorphic to a ball is closely related to the notion of its surface being homeomorphic to a sphere.

This algorithm as written will also check whether each vertex is homeomorphic to a 0-ball. We define the operation ($V \approx$ 0-◯) to mean that the grid vertex V

does not lie in a region of space where the surface mesh is inside-out. In other words, as long as the surface mesh is properly embedded in space at grid point V, then $(V \approx 0\text{-}\bigcirc) = \texttt{true}$.

If this homeomorphism test at line 8 fails, then we find every cubic cell Q whose boundary contains C, and we label the 3-cell to be re-sampled by setting $\mathrm{L}(Q)$ to `true`.

We then run through all 3-cells in the grid at line 12 of Figure 40. If the intersection created by any 3-cell is not homeomorphic to a 3-ball, or if any of its boundary facets failed their homeomorphic tests (denoted by $\mathrm{L}(C)$=`true`), then we re-sample this 3-cell. This re-sampling procedure is performed at line 14, by replacing S with the union of itself and the convex hull H() of the the solid intersection between the C and S.

As presented, this algorithm essentially takes the solid intersection between the surface mesh and every cell in the grid. It then checks this intersection to see if it is homeomorphic to a ball. If anything is not homeomorphic to a ball, then the surface inside this cubic grid cell is replaced by the convex hull of its intersection with S. Keep in mind that the algorithm in Figure 40 describes an idealized, unoptimized algorithm that requires the computation of every convex hull of every d-cell in the grid. We discussed some shortcuts that can be taken in Section 7.5.2.

If, instead of only performing these operations with the grid G, we performed the algorithm with *all possible translations and rotations* of G, then we would have an algorithm very similar to a constant radius filleting [153]. Constant radius filleting uses similar morphological operations with all balls of a constant radius, while our method uses a subset of all cubes of a constant size. From this perspective, we can claim that the algorithm presented in this chapter approximates a morphological closing of the input surface mesh.

This morphological point of view makes sense when applied to a fluid simulation

— we must filter out all topological features which are smaller than a grid cell in order to ensure that the topology of our simulation matches the topology of our surface. Because our algorithm is designed to work within a fluid simulator, we prefer to use the cell complex G defined by the dual of our simulation mesh for these morphological operations. Fluid simulation data is transferred across cell faces, edges, and corners, so the lower dimensional facets of the dual cell complex G represent the methods of communication across fluid cells. If we ensure that the surface topology is simple along the boundaries of G, we make the topology of the simulation grid match that of the surface mesh.

By defining G as the dual to the simulation grid, we can also generalize our method to work with any simulation grid. In particular, fluid simulation methods using tetrahedral meshes [78, 8] or Voronoi cell complexes [133, 25] should be able to use a variant on the algorithm described in this chapter.

7.5.5 Topological Control

The method presented in Section 7.5.3 will preserve all thin sheets and spindles unless there are significant self-intersections. In case we desire different behaviors, we can modify the presented algorithm by altering our definition of topological validity. For example, we could force the breakup of thin liquid spindles by declaring that a cell face is invalid when the surface intersects the face but not its bounding edges. We can quickly find such faces by searching for cells that overlap the same triangles but do not have any edge intersections on the boundary of their common face. To create a re-sampled surface that respects this altered definition of topological validity, we can simply remove any vertices on that face from the input to the convex hull algorithm. We can similarly remove many vertices from the convex hull input if we want to speed up computation time, although we did not find this optimization necessary.

An alternative to modifying the definition of topological validity is to manually

perform topological changes to the surface through explicit mesh surgery. We have found that this approach for cutting thin spindles of liquid produces better results than the method outlined in the preceding paragraph, because explicit cutting allows us to have tighter control over the thickness of the surface before the separation. We can detect thin spindles of liquid while performing the local surface maintenance operations (Section 7.4), when we flag edges that will produce non-manifold geometry if collapsed. Of these non-manifold cases, we can easily identify any thin spindles by their triangular cross-section. To perform the mesh surgery, we cut the mesh at the triangular cross-section, seal each end with a new triangle, and perturb the two new strands away from each other by a small offset. A similar operation is explained in detail by Lachaud et al. [82].

As explained in Section 7.2.1, thin spindles of liquid consistently lead to topological changes in real-world liquids. Due to a surface-tension-based Rayleigh-Plateau instability, the liquid spindle rapidly collapses until it is extremely thin and then breaks in two. Our explicit cutting operation precisely mimics this behavior in the discrete setting by separating the thinnest possible unit of a discrete surface. This cutting method is also quite efficient — it is detected for free during standard surface maintenance operations, and it requires about as much work as a single edge collapse.

Both of these proposed methods give us control over the breakup of thin liquid spindles, agreeing with the instability of thin spindles due to Rayleigh-Plateau instability suggested in Section 7.2.1. Using these proposed topology modifications, the animator can specify either the topological grid length or the maximum allowed edge length in order to force the breakup of thin liquid spindles.

7.5.5.1 Revisiting the Stability of Geometric Structures in Two-Phase Flow

We will now re-visit each of the eight cases mentioned in Section 7.2.4 and check whether our topological modification algorithm reproduces the physically appropriate behaviors. If our algorithm is physically accurate, it should preserve any structure that was deemed physically stable. On the other hand, any structure that is topologically unstable will naturally degenerate into a different small geometric structure. Our algorithm should delete such unstable configurations and replace them with physically plausible stable geometry.

Volume-like liquid: We decided that this structure should be stable in Section 7.2.4. Indeed, the intersection of a huge liquid object with a small cubic cell will yield the cell itself. The cell is topologically valid (in fact it is identical to its convex hull), so our algorithm will preserve this stable structure.

Volume-like gas: Section 7.2.4 declares that this structure should be stable as well. The solid intersection of a huge void with a cubic cell yields the empty set. No geometry is left inside of this cell, so the geometry is trivially topologically valid. Our algorithm will preserve this stable structure.

Sheet-like liquid: As discussed in Section 7.2.4, a thin sheet of liquid should be stable. Its solid intersection with a cubic cell will create one component of geometry, and the intersections with the cell walls will either be a single solid component or no component at all. The intersections with cell edges will either be absent or a single, properly oriented line segment. Everything about this structure is topologically valid, so our algorithm will preserve thin liquid sheets.

Sheet-like gas: in contrast to thin liquid sheets, we decided that thin sheets of air should be *unstable* in Section 7.2.4. The solid intersection (where the liquid phase represents the solid and the gas phase represents a void) between a thin sheet of air and a solid cubic cell yields two disconnected components separated by a thin sheet of void. This resulting surface is not homeomorphic to a 2-sphere, and many of its

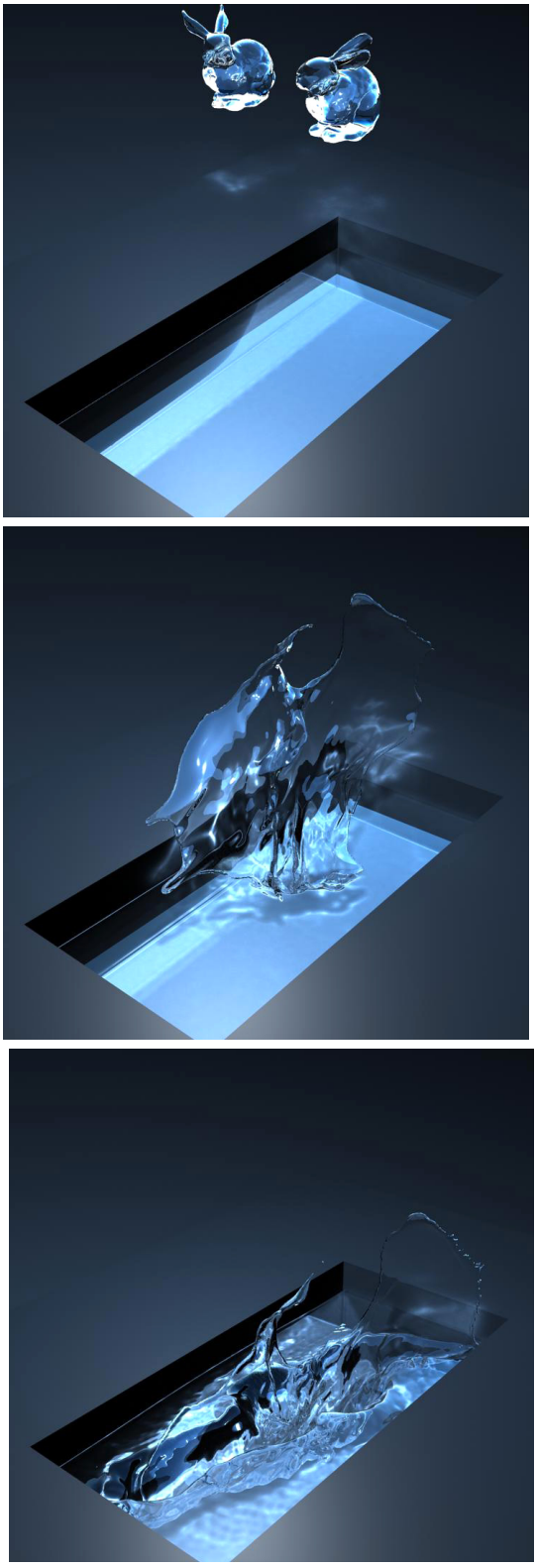


Figure 41: Two bunnies splatter into each other, producing a thin liquid sheet that merges with the pool of water below.

intersections with cell faces and edges have surfaces which are not homeomorphic to 1-spheres and 0-spheres. This geometry is topologically *invalid*, and our algorithm will replace it with its convex hull. The convex hull essentially merges together the two liquid components and deletes the sheet of air.

Although our algorithm correctly merges together the two components of liquid, it somewhat nonphysically deletes all traces of the air component. In reality, the sheet of air would break up into tiny bubbles. However, if we created small bubbles within a fluid cell in our simulation, they would not behave independently because we do not have a high enough computational resolution within a single cell to resolve several separate fluid phases. Because adding bubbles dramatically increases the required computational resources to simulate a simple merging fluid flow, we cannot afford to insert them into our simulation. Furthermore, because these bubble components have such low mass, they do not contribute nearly as much to the resulting fluid dynamics. As a result, we found it acceptable to simply omit the tiny bubbles that would normally result from a disintegrating sheet of air.

Thread-like liquid: We decided that this structure should be unstable in Section 7.2.4, due to a Rayleigh-Plateau instability. The intersection of a thin thread of liquid and a cubic cell yields a clipped thread of liquid. This structure is topologically valid, because its surface is a single component with no holes or voids, and any intersections with the cell boundary are topologically valid as well. If we allowed this geometry to persist, then thin strands of liquid would never break into smaller droplets. We most likely do in fact want these threads to break up, so we discussed two methods for splitting them apart as a post-process in Section 7.5.5. After explicitly identifying these strands and cutting them at their thinnest regions, these thin threads of liquid will properly degenerate into more stable geometry like small droplets and large volumetric structures.

Thread-like gas: This structure was labeled unstable in Section 7.2.4, due to a

Rayleigh-Plateau instability. If we take a large body of liquid with a thin thread of air inside it, and we intersect this structure with a cubic cell, then we are left with a liquid cube with a small thread subtracted from it. This surface of this structure is homeomorphic to a torus (not a sphere), so it is topologically invalid. We replace this structure with its convex hull, which essentially closes the hole in the torus and deletes all surfaces from this cell (making it completely filled with liquid). As with the sheet-like structure of air, we neglect any resulting bubbles from this collapsing tube of air in order to keep the subsequent fluid simulation feasible.

Point-like liquid: According to our reasoning in Section 7.2.4, small droplets of water are *stable*. The solid intersection between a tiny ball and a cubic cell is just the ball itself. The surface of this ball is homeomorphic to a sphere, and the surface of its intersections with the cell walls and edges (if it happens to intersect the cell boundary) will be homeomorphic to lower dimensional spheres as well. This structure is topologically valid, so we preserve it.

Point-like gas: Lastly, we decided that small bubbles were stable in Section 7.2.4. If we construct a large body of liquid, subtract a tiny bubble of air from it, and intersect this whole structure with a cubic cell, then we are left with a cube with a void in the center. In case the bubble intersects the wall of a cell, then the void will be located on one of the cell faces instead. Unfortunately, this structure is topologically invalid and will be replaced by its convex hull, essentially deleting the air bubble.

While this behavior is unphysical, the problem is not straightforward to solve without adding additional complexity to the fluid simulation. As discussed in the "sheet-like gas" case, we need to add additional degrees of freedom to the simulation in order to resolve multiple fluid phases within a single fluid cell. Several particle-style methods seem promising ([56], [61], [93], and [74], for example), though we have not implemented any small air bubble behavior in our simulator yet.

7.6 Sewing Meshes Together

Before re-sampling the surfaces in Section 7.5.3, we subdivide the original surface mesh so that it perfectly lines up with the boundaries of each invalid cell. We first subdivide triangles where they intersect the boundary edge of an invalid cell, and then we subdivide edges where they intersect the boundary faces of an invalid cell, following the algorithm of Du et al. [42]. Next, we discard the original surface within the invalid cells and replace them with topologically valid convex hull surfaces from Section 7.5.3. Finally, we must stitch the surfaces together along the cell boundaries.

7.6.1 Subdivision Stitching

One possible mesh-stitching strategy is to progressively collapse triangles on both sides of the cell boundary until the detailed intersection between the surface and cell face is reduced to a line segment. However, such a strategy deletes important Lagrangian surface samples and can lead to robustness problems (see Section 6.3.2 and [154]). Instead of altering the original surface and destroying its Lagrangian data, we choose to only alter the new re-sampled surface.

Initially, if the new triangles output from the convex hull already perfectly match up with the triangles on the other side of the face (the piecewise linear curve formed by the triangle edges coincident with each boundary face is identical for both the original and new surfaces), then we are finished sewing the surfaces together at this

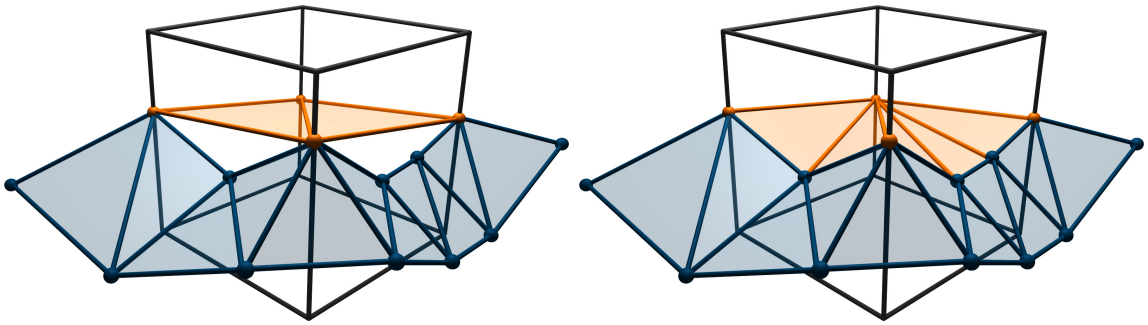


Figure 42: Before and after sewing the meshes together.

face. Otherwise, we have to subdivide the new surface until its boundary curve matches that of the original surface. First, we limit the number of cases that we need to address by ensuring that each new triangle shares an edge with at most one triangle outside of the invalid cell region — for each new triangle that shares edges with two or more triangles outside of the invalid region, we place a vertex at its barycenter and split it into three new triangles. Next, for each new triangle that does not perfectly match up with a segment of the boundary curve, we subdivide it in two by adding a point on its boundary edge and snapping it to one of the vertices on the boundary curve. We then recursively subdivide each of these newly-created triangles in the same way until the curves perfectly match (each curve eventually reduces to a single line segment in the base case), similar to the algorithm used by Bischoff and Kobbelt [14] to stitch together NURBS surfaces. Our algorithm is guaranteed to terminate, and it is linear in the number of vertices coincident with the face. After this step, the newly-resampled mesh surface from Section 7.5.3 will be properly sewn together with the original surface along its boundary, creating a closed, oriented, manifold surface mesh. See Figure 42 for a visual aid.

7.6.2 Smooth Surface Interpolation

Both marching cubes-style lookup tables as well as our convex hull algorithm in Section 7.5.3 use piecewise linear surfaces to connect vertices together. These low-order connecting surfaces are indistinguishable from the original surface mesh when the triangles in the surface mesh are about the same size as a fluid grid cell. However, our method allows the surface mesh to have a much higher resolution than the simulation grid, so the triangles output by our convex hull will often seem disproportionately large. To maintain a detailed surface with many small triangles, we repeatedly subdivide these large triangles until their edges are as short as the maximum edge length allowed in our surface mesh. These subdivisions create new vertices, and we are free

Figure 43: Two streams of water collide, filling a domain with liquid.

to place them wherever we like.

The first method we tried for relocating these vertices was a surface optimization scheme. We created a smooth surface by minimizing the discrete bi-Laplacian evaluated at each of the vertices [16]. Unfortunately the resulting system is numerically very poorly conditioned and took an unnecessarily long amount of time to converge on meshes with arbitrarily-shaped triangles.

Instead, we decided to use butterfly subdivision when placing our new vertices, similar to Brochu and Bridson [23]. We found that the modified scheme of Zorin et al. [160] worked best, because a significant portion of the vertices in our simulations did not have a valence of six. After applying this interpolating subdivision to the triangles output by our topological correction algorithm, the surfaces exhibit both correct topology and high degrees of smoothness.

7.7 Input Parameters

Like the algorithm in Chapter 6, this chapter has relatively few parameters to set. The topological detection depends on the grid size, which is identical to the fluid grid resolution in this chapter. We are free to change the topological resolution if we like, but then we lose any guarantees about topological validity if the topological grid is not the dual of the fluid simulation grid. The main tool for fixing topology in this chapter is the convex hull, which does not depend on any input parameters.

The topological algorithm is unconditionally stable, so it does not depend on any minimum time step size. Some degenerate configurations like completely flat shapes can cause errors when sewing together the convex hull to the rest of the surface, but this problem can be solved by jittering vertices or by more intelligent collision resolution.

7.8 Results and Discussion

Our simulations all show different combinations of thin sheets, strands, and droplets due to a high resolution surface mesh being advected through a lower resolution fluid simulation. Figure 43 shows two streams of water colliding to fill up a domain. Most other simulation techniques would be unable to resolve such a continuous stream of thin sheets, but ours easily avoids the deletion of the large thin structures. The simulation took between 0.5 and 10 seconds per frame, and it finished with 800k triangles in a $100\times50\times100$ fluid domain. Figure 44 shows a comparison between our method and a level set surface tracker on a 60^3 domain. Our method clearly resolves more surface details and thinner sheets, although our method is considerably more expensive due to the mesh maintenance. This example took about 8 seconds per frame and finished with 200k surface triangles. Figure 41 shows two bunny meshes creating a thin liquid sheet after violently colliding into each other. The thin sheet then merges with a larger body of water below. This example averaged 28 seconds per frame for a

60×120×120 fluid simulation with at most 520k triangles. Figure 36 shows how our method for persistent liquid sheets combines with a mesh-based surface tension technique (explained in detail in Chapter 8 [145]) in order to simulate the realistic breakup of a thin liquid film. The domain is 160×80×160 (although most of it is empty), and it took 1 to 12 seconds per frame to simulate 320k triangles. Finally, Figure 35 shows an example of a hallway flooding. This simulation was able to simulate thin sheets and droplets despite its coarse fluid simulation resolution. It simulated at 2 to 13 seconds per frame and finished with 600k triangles in a 120×60×60 fluid domain. Our simulations typically output one frame of animation for each time step in the simulation.

Because we use fairly low resolution fluid simulations, and because the thin sheets in our videos do not occupy the full volume of the fluid simulation, we found that the performance of our method is more closely tied to the resolution of the surface mesh than to the fluid grid. The majority of the computation time in our examples was spent on mesh-based operations (subdivision and edge collapses, advection of the surface vertices, topology changes, and mesh-based surface tension calculations), although the expense of the surface tracker becomes negligible with higher resolution

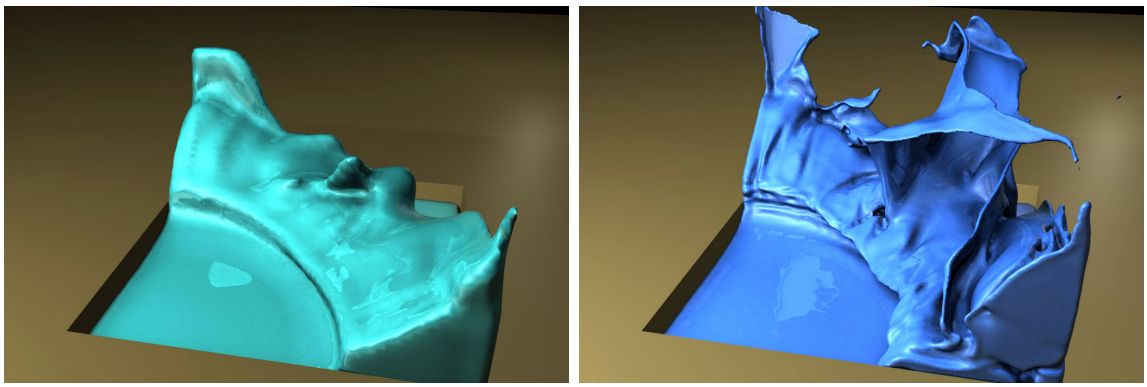


Figure 44: Comparison between a level set surface tracker with 2nd order advection (left) and our method (right) on a 60^3 grid.

volumetric fluid simulations and coarser surface meshes. The fraction of total simulation time spent in the topological change algorithm varied from 17% to 57% for the examples in this chapter (Figure 36 and Figure 43, respectively). The signed distance calculation is by far the most expensive part of the topology change algorithm, followed by a global recomputation of the triangle mesh connectivity data structure after any mesh surgery. We believe that we can significantly speed up our implementation in the future by using a hardware voxelization routine and only locally updating mesh connectivity.

While grid-based approaches have one dominant parameter (grid resolution), our method has two: topological resolution (grid cell size) and surface detail resolution (triangle edge length). These independent resolution parameters allow our method to track hundreds of surface vertices within a single grid cell while maintaining the same topology as the underlying fluid simulation. The cost of maintaining extra vertices in a single grid cell is similar to that of a particle level set [47], though a particle level set with 64 particles per cell will resolve significantly fewer surface details than our method with 64 surface vertices per cell.

The cost of computing topological changes in our method is modest, though its expense grows linearly with surface detail resolution. For example, if our surface detail resolution matches the topological change resolution, then our method will behave similar to that of Müller [98] (near real-time performance), with significantly reduced diffusion of surface details. As the surface resolution increases, so does the cost of advecting the mesh and computing topological changes. The memory consumption of our method scales linearly with the surface resolution, in order to store the surface mesh and the grid cells that intersect it.

Figure 45 illustrates a failure mode of common surface trackers with a low fluid resolution. In this scenario, grid-based implicit surface trackers (level sets, particle level sets, and semi-Lagrangian contouring, for example) limit the surface detail to

that of the grid resolution, which causes catastrophic deletion of thin features. Naively increasing the surface resolution can cause the topology of the fluid simulation to differ from that of the surface tracker. Such mismatched topologies can lead to artifacts such as several surface components floating within a single fluid cell, as well as an unphysical breakup of thin sheets. Artificially diffusing the level set [75] can force these floating components to merge together, at the expense of limiting surface detail, losing volume, and punching holes in thin sheets. Our method is the only surface tracker to date that can showcase arbitrarily high levels of surface detail without sacrificing topological agreement between the surface and the fluid.

Although we did not use exact arithmetic to compute the topological details of our algorithm, we did take precautions to make it robust in the face of numerical precision errors. In the event that a numerical error causes impossible topology or leads to difficulty sewing together the mesh, we mark all cells near the error as topologically invalid and re-sample the geometry within them.

We tried to preserve as much Lagrangian surface information as possible throughout the surface tracking process, and our topological constraints allow us to advect surface meshes that are much higher resolution than the fluid velocity field. Consequently, we can animate highly detailed surfaces through time without losing surface features to a re-sampling process. However, this retention of high frequency details may not always be desirable. Some possible ways to smooth out extra details are with geometric smoothing or simulated surface tension. In addition, our topological corrections will often replace high-frequency surface details with a less detailed surface whenever they detect invalid geometry. This replacement can lead to noticeable temporal discontinuities when simulating with an extremely low resolution fluid simulation grid.

Because our topological algorithm primarily merges fluid regions together and rarely splits things apart, the liquid in our simulations tends to exhibit an overall gain

in volume, especially after long, violently splashing simulations with many topological merges. This is the opposite of the common volume-loss problem with level set-style surface trackers, and the amount of volume gain is closely tied to the coarseness of the fluid grid.

Nevertheless, we intentionally used low-resolution fluid simulations in many of our examples to show the advantage of separating the visible surface resolution from the resolution of the velocity field. The hallway in Figure 35, for example, is only 30 grid cells wide. Despite the coarse physics, our method is able to develop many high resolution surface details while still matching the topology of the low resolution fluid simulation. The additional detail gained from separating the fluid simulation from its surface allows us to avoid simulating expensive high resolution physics when only a highly detailed surface is necessary.

7.9 Conclusion and Future Work

We have presented a mesh-based surface tracking method for fluid animation that both preserves fine surface details and robustly adjusts the topology of the surface in the presence of thin features like sheets and strands. Our local convex-hull-based method for correcting the topology of surfaces enforces the overall topological behavior of real-world liquids, and we can achieve arbitrarily thin sheets and strands without re-sampling important Lagrangian surface details. We further reduce re-sampling artifacts with a subdivision-based mesh-stitching algorithm, and we use a higher order interpolating subdivision scheme to determine the location of any newly-created vertices. We have shown how our method can be applied to efficiently produce high-quality fluid surface animations, even when coupled to under-resolved fluid simulations.

In the future, we would like to look into mesh-based methods for creating sub-grid scale physical behaviors. In addition to using a mesh-based surface tension method

(Chapter 8 [145]), we believe that other surface-focused approximations to forces in the Navier-Stokes equations like the vortex sheet method of Kim et al. [75] could yield highly detailed simulations without the expense of running a full fluid simulation. Lastly, one potentially fruitful area of future work is to investigate the possibility of topologically separating fluid surfaces instead of merging them together, as done for elastic models by Teran et al. [141] and Nesme et al. [106].

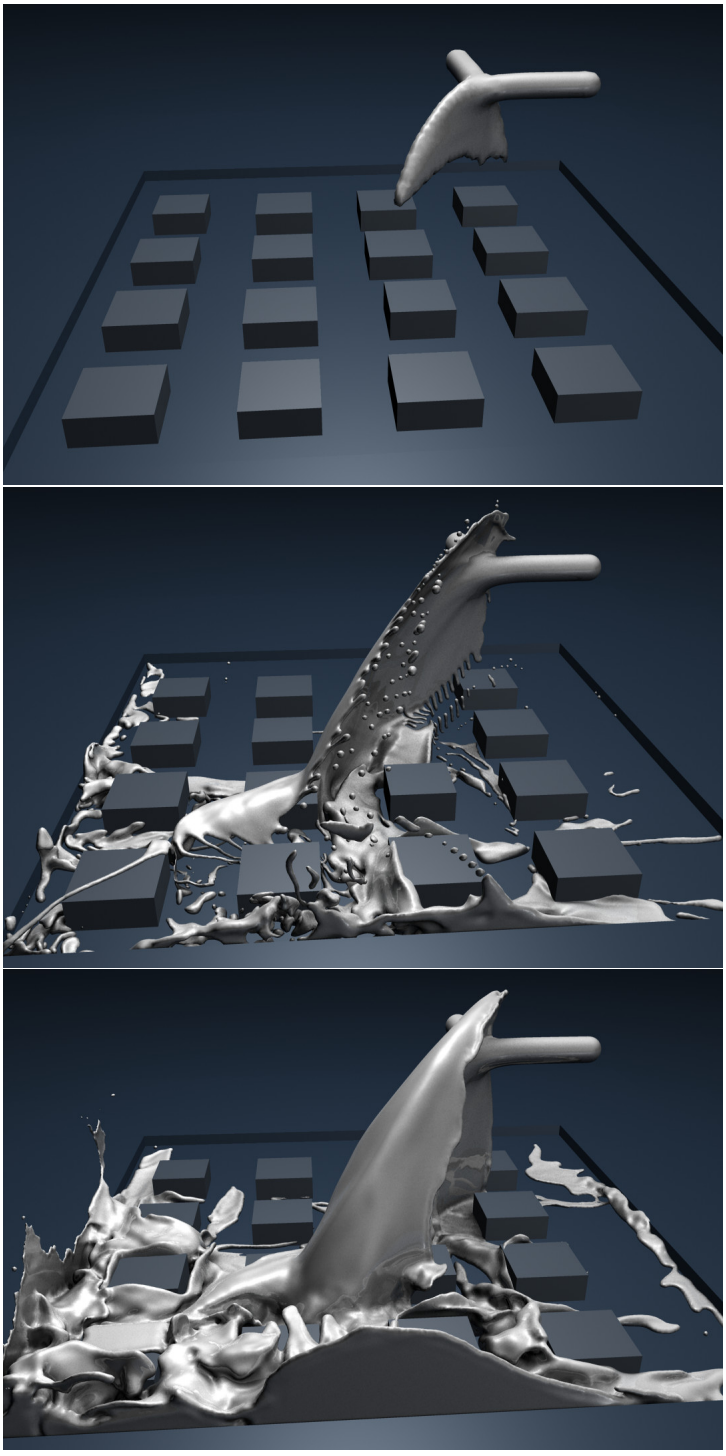


Figure 45: In this example, the level set method (left) deletes thin sheets before they even touch the ground. The method from Chapter 6 (middle) re-samples the surface from a grid during topological changes, so it cannot merge together multiple thin sheets without deleting large portions of the surface. The method of this chapter (right) merges together thin sheets without rampant thin feature deletion.

CHAPTER VIII

LAYERED SURFACE TENSION SIMULATION

Surface tension forces are responsible for many phenomena that contribute essential details to liquids and interfaces in nature. The formation of a water droplet can be attributed to a growing surface tension instability, and small ripples on the surface of a liquid are primarily driven by surface tension. In the typical Eulerian solvers used for computer animation, surface tension effects are often neglected, and effects such as droplet pinch off occur purely as a result of insufficient computational resolution. Once tension effects are explicitly modeled with forces at the liquid interface, many cells are required to resolve the shape of a single droplet. In addition, surface tension forces impose a strict time step restriction on the solver. These requirements of a high resolution and small time steps make the simulation of large bodies of liquid infeasibly expensive.

In this chapter, we present an algorithm for efficiently and robustly simulating surface tension effects that is based on a triangle mesh discretization of the fluid surface, as described in Chapter 4. The mesh-based surface tension method is decoupled from the resolution of the Eulerian grid and allows for an efficient simulation of sub-grid scale surface tension effects, including droplet pinch off and surface tension waves. At the same time, it allows us to separate the small scales of detail that are best simulated on the surface from the larger scales that can be resolved by the Eulerian fluid simulation grid.With this triangle mesh-based surface representation, we can use a robust finite element method to discretize our surface tension forces. In particular, because surface tension forces depend primarily on surface curvature, we can take advantage of the large body of discrete differential geometry literature involved in accurately

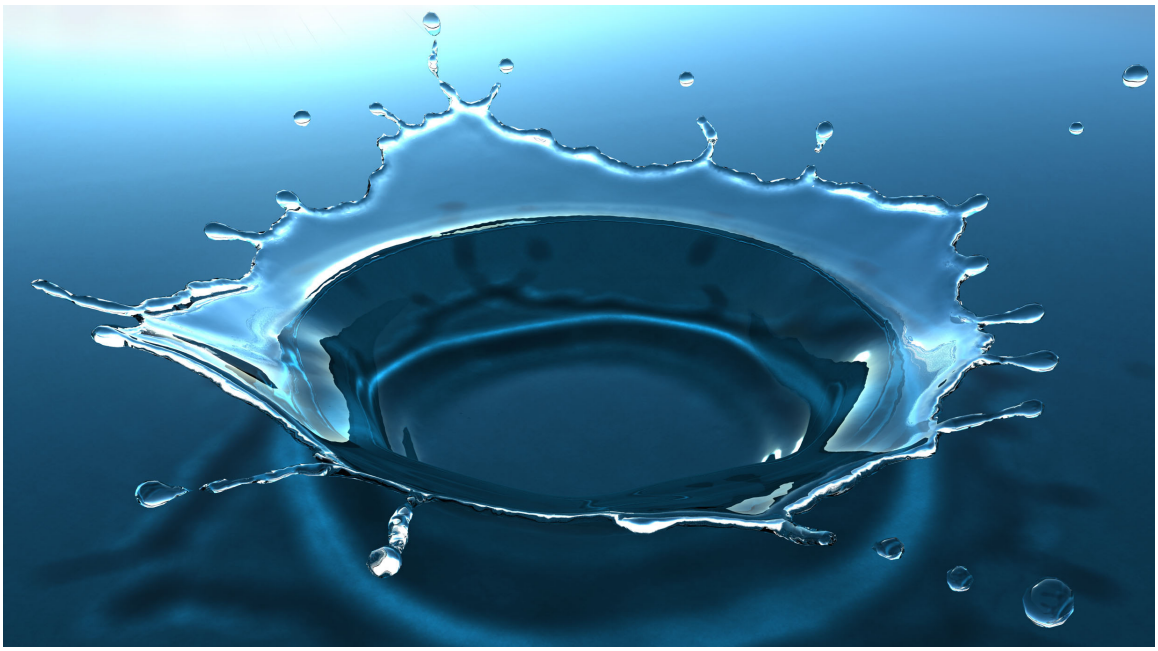


Figure 46: Our method allows us to efficiently simulate complex surface tension phenomena such as this crown splash. The small scales are handled with our surface approach, while the larger scales are computed with the Eulerian simulation. For the shown simulation, our method requires only 22.3 seconds per frame on average.

computing the curvature of a surface. The mesh-based discretization also allows us to guarantee a conservation of mass throughout our simulation, which directly causes physically realistic secondary effects like bulging surfaces and Rayleigh-Plateau instabilities (see Chapter 7).

The key attributes of this chapter's approach to surface tension are:

- A novel method for efficiently simulating sub-grid surface tension effects that computes wave dynamics on the discretized fluid surface.
- An algorithm that has a significantly relaxed time step restriction in comparison to previous approaches in graphics.
- The efficient and robust handling of droplet pinch off as well as merging effects with local volume preservation on sub-grid scales.
- A simple and efficient non-oscillatory approximation to sub-grid surface tension using mesh-based volume-preserving mean curvature flow.

These contributions combine to produce highly detailed animations of surface tension phenomena with subtle secondary effects at a fraction of the cost of previous methods.

We will now briefly outline the different steps of our simulation algorithm, which are also illustrated in Figure 47. For simulating the fluid, we use the standard Eulerian fluid solver with an embedded surface mesh from Chapter 4. The input to our algorithm is a triangle mesh F representing the liquid interface. First, we advect F through the velocity field given by the previous step in the fluid simulation, using any of the advection methods described in previous chapters.

Next, we compute sub-grid cell surface tension dynamics by solving a wave equation on the surface. To set up the wave equation, we first filter out the surface features of F that are too high resolution to be captured by the fluid grid, in order to create a low-resolution smooth surface, S. We then initialize the wave equation problem with wave heights equal to the difference vectors between points on F and S. We solve the wave equation on F using an implicit Newmark scheme, resulting in surface dynamics that make F ripple and oscillate about the smoother surface S.

After that, because S has a low enough curvature to be fully resolved on the fluid grid, we use S to compute Eulerian surface tension forces in the fluid simulation. We show that surface tension forces can be treated as the time derivative of volume-preserving mean curvature flows in order to allow for much larger time steps than with other explicit schemes. We run a volume-preserving mean curvature flow on S proportional to the time step size and surface tension strength, producing another new surface, T. The difference vectors between T and S give us the surface tension forces that can be included as boundary conditions for the Eulerian pressure solve. Finally, we solve this system to get our final divergence-free velocity field and then move onto the next time step.

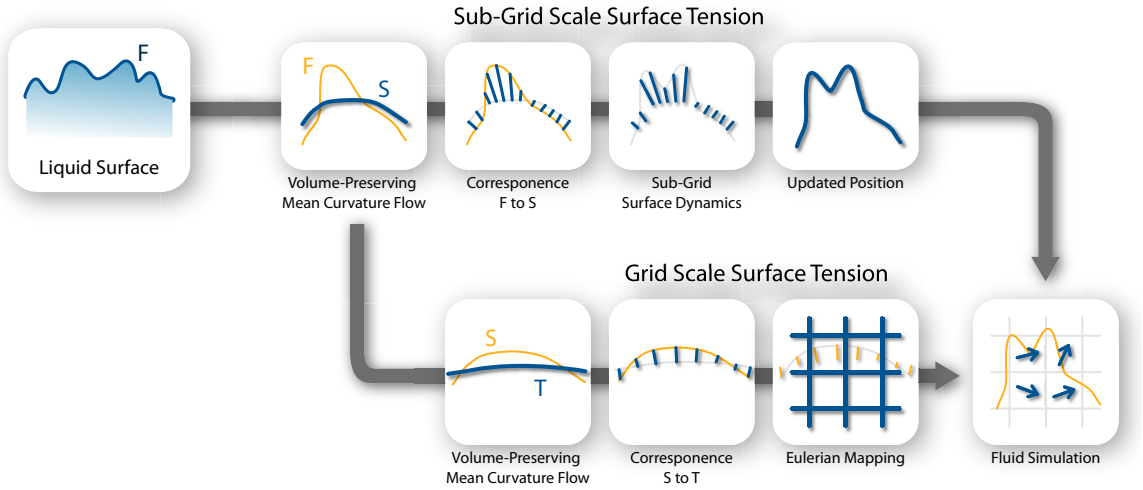


Figure 47: Overview of our surface tension approach. The two rows illustrate the steps performed for the sub-grid surface tension dynamics in the top row, and the Eulerian surface tension in the bottom row.

8.1 Fluids with Surface Tension

As mentioned in Chapter 4, the dynamics of an incompressible viscous fluid can be described by the Navier-Stokes equations (Equations 9 and 10). In this chapter, we also consider surface tension forces, so we add an additional term to the right of the momentum equation, yielding our updated Navier-Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{g} + \sigma\kappa \tag{15}$$

$$\nabla \cdot \mathbf{u} = 0 \ . \tag{16}$$

To review, $\mathbf{u}$ is the velocity of the fluid, ρ is the fluid density, p is the pressure, ν is the viscosity, and $\mathbf{g}$ is the external force. We have added the surface tension term $\sigma\kappa$, which creates a force proportional to the surface curvature κ and the surface tension coefficient σ. Note that the surface tension forces are zero everywhere except at the liquid-gas interface. The Navier-Stokes equations are typically discretized on a Eulerian grid with cell size Δx and a time step of Δt, using a staggered MAC grid for storing the velocity information (Chapter 4).

For level-set based surface representations, an established way to implement surface tension effects is to compute κ with the Laplacian of the signed distance function. Because the surface tension forces act as a pressure jump across the interface, they can be added as boundary conditions to the pressure solve [68, 60]. This approach gives stable results for a moderate range of surface tension strengths. To improve stability, [137, 138] propose a method to compute the $\sigma\kappa$ term not directly from the level-set, but by first computing a volume-preserving mean curvature flow (VCF) of the liquid interface. This method computes surface tension froces by taking the temporal finite difference between the original surface and the one experiencing curvature flow. This curvature-flow based strategy leads to significantly more stable simulations of surface tension flows, because it offloads many of the numerical calculations into a much more stable and well-studied problem (that of mean curvature flow).

Unfortunately, the methods above completely rely on the Eulerian grid in order to resolve surface features. This can cause severe problems when surface features reach small sizes on the order of a grid cell wide. These small features violate the fundamental assumption that all features are large enough to be adequately sampled by finite difference stencils. The finite difference formula for computing surface curvature κ requires surface features to be at least three grid cells wide. Not only do small features cause this curvature metric to become inaccurate, but they can create wildly unstable behaviors as well. For example, a direct evaluation of curvature from the level set can lead to noticeable ghost forces and random accelerations of small droplets. Furthermore, any level-set based method can unphysically delete small surface components, further contributing to the inaccurate computation of surface tension forces. A more thorough discussion of some shortcomings of level-set based surface trackers can be found in Chapters 6 and 7.

As noted throughout the surface tension simulation literature [137, 145], surface

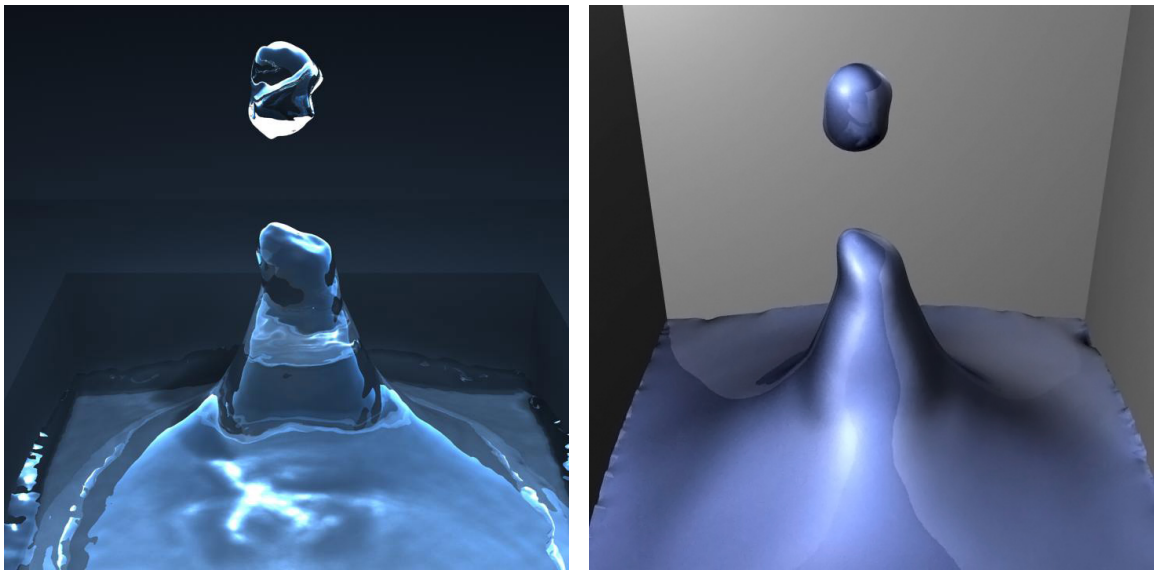


Figure 48: This image shows a fluid simulation featuring a splash with a detaching droplet of water. The visible surface F is shown on the left, and the smoother surface S is shown on the right. The sub-grid surface dynamics are computed with respect to the surface on the right.

tension forces require a small time step $\Delta t \leq \Delta x^{3/2}$ in order to simulate them explicitly. This stringent restriction has led to algorithms that perform multiple computational steps for the surface tension during a single step of the fluid simulation. Implicit methods [59] predict the curvature of the surface for the next time step, and sub-cycling methods [35] perform additional advection operations to evolve the surface based on surface tension forces. As mentioned earlier, Sussman and Ohta [137, 138] transfer many of the instabilities of the surface tension integration into a different problem, that of computing a mean curvature flow. We use a similar method for computing Eulerian surface tension forces, though we use an explicit surface mesh instead of an implicit level set surface. We also simulate sub-grid cell surface dynamics with an implicit, symplectic wave equation integrator, which means that we can simulate capillary waves at arbitrarily large time steps while conserving energy.

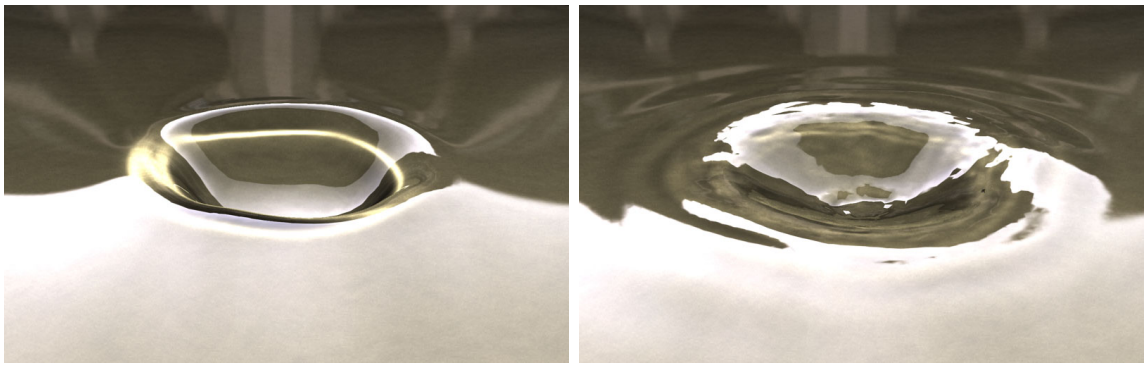


Figure 49: This image shows a liquid surface after a drop impact. A simulation without sub-grid dynamics is shown on the left, while the right image demonstrates how the method in this chapter can simulate additional small scales waves at the surface.

8.2 Surface Tension as a Time-Derivative of Mean Curvature Flow

As explained earlier, we use a variant of the method of Sussman and Ohta [137, 138], which sidesteps computational problems associated with surface tension forces by converting them into computational problems associated with mean curvature flows. In this section, we will explain why it is acceptable to compute surface tension forces in this manner, and we will elaborate on the implications of this mathematical technique.

Suppose we have a surface S experiencing surface tension, so it evolves according to the differtial equation

$$\frac{\partial^2 S(t)}{\partial t^2} = \sigma \nabla^2 S(t) \tag{17}$$

where t is time and σ is a constant representing the surface tension strength. The initial conditions of the flow are defined by the surface $S(t = t_0) = S_0$. (Note that we are using a one-dimensional form of this equation with the Laplacian $\nabla^2 S$ instead of the curvature κ for ease of explanation. Rest assured that the results still hold for surface curvatures in any dimension.)

In addition to Equation 17, suppose we have a different surface C undergoing

mean curvature flow:

$$\frac{\partial C(t)}{\partial t} = k\nabla^2 C(t) \tag{18}$$

where k is a constant representing the strength of the curvature flow. We can design the curvature flow such that $k = \sigma$ and its initial surface is identical to that of the surface tension flow $C(t = t_0) = S_0$.

Next, let us examine how the flows differ as we deviate from time $t = t_0$ via a Taylor series expansion. We can consider any point in time t as a sum of the initial time t_0 and a discrete time step Δt, so $t = t_0 + \Delta t$.

$$\frac{\partial^2 S(t)}{\partial t^2} = \sigma\nabla^2 S(t) = \sigma\nabla^2 \left(S_0 + \frac{\partial S(t_0)}{\partial t}\Delta t + O(\Delta t^2) \right) \tag{19}$$

$$\frac{\partial C(t)}{\partial t} = \sigma\nabla^2 C(t) = \sigma\nabla^2 \left(S_0 + \frac{\partial C(t_0)}{\partial t}\Delta t + O(\Delta t^2) \right) \tag{20}$$

We can subtract the second equation from the first

$$\frac{\partial^2 S(t)}{\partial t^2} - \frac{\partial C(t)}{\partial t} = \sigma\nabla^2 \left(\frac{\partial S(t_0)}{\partial t}\Delta t - \frac{\partial C(t_0)}{\partial t}\Delta t + O(\Delta t^2) \right) \tag{21}$$

and rearrange terms to find

$$\frac{\partial^2 S(t)}{\partial t^2} = \frac{\partial C(t)}{\partial t} + O(\Delta t). \tag{22}$$

Equation 22 basically tells us that the time derivative of a mean curvature flow integrated across a time step Δt is a valid approximation for surface tension behavior. Sussman and Ohta [137, 138] used this idea to create a stable and efficient numerical method for surface tension behavior. To approximate the surface tension forces at a point on the surface $S(t)$, they force it to undergo mean curvature flow for one time step Δt and arrive at a new surface $C(t+\Delta t)$. Then the first order forward difference between these two surfaces $\frac{C(t+\Delta t)-S(t)}{\Delta t}$ yields the acceleration of each point on $S(t)$ due to surface tension. Higher order finite differences can be used with the mean curvature flows, though the accuracy will be ultimately limited by the $O(\Delta t)$ term in Equation 22.

At this point, we may question whether this is a worthwhile strategy at all. In fact, we may actually be making things worse for ourselves by converting the surface tension problem into a mean curvature flow problem. As mentioned earlier, the time step required to explicitly integrate Equation 17 without running into numerical instabilities is $\Delta t \leq \Delta x^{3/2}$. In contrast, the time step required to explicitly integrate Equation 18 is $\Delta t \leq \Delta x^2$. Clearly, the mean curvature flow problem requires an even *smaller* time step than the surface tension problem. However, the mean curvature problem is still easier to deal with for three reasons: implicit integration, increased surface tension stability, and divergence-free sub-cycling.

Implicit integration: Mean curvature flows are well-studied throughout the scientific literature. Within the field of computer graphics, mean curvature flows are widely used to create smooth surfaces [139, 39]. This flow problem is also closely related to the viscosity term in the Navier-Stokes equations (Equation 15) and the equation used to model heat flow in an environment and diffuse chemicals in reaction-diffusion systems [148]. To get around harsh stability restrictions in all of these applications, researchers use implicit integration techniques [39, 30]. Implicit integration of equation 18 provides *unconditional* stability of the system by converting it into a simple linear system, making this mean curvature flow problem quite convenient in practice.

Increased surface tension stability: The stability requirement for explicitly simulating Equation 17 is

$$\Delta t \leq \sqrt{\frac{\rho_L + \rho_G}{(2\pi)^3 \sigma}} \Delta x^{3/2} \tag{23}$$

where Δt is the simulation time step, ρ_L is the density of the liquid phase, ρ_G is the density of the gas phase, σ is the surface tension strength, and Δx is the size of a grid cell or triangle edge, depending on your discretization of space. Sussman and Ohta [137] drastically improve upon this stability requirement by reinterpreting the surface tension term as a finite difference between mean curvature flows. Ignoring the

stability requirements of the mean curvature flow, the stability requirement for their new surface tension integration is

$$\Delta t \leq \frac{\Delta x(\rho_L + \rho_G)}{2\pi}. \tag{24}$$

This result is an obvious improvement, especially for detailed simulations with small Δx.

Divergence-free sub-cycling: The surface tension forces in Equation 15 will often be the most numerically restrictive forces in the system, particularly in the presence of a large surface tension coefficient σ. As a result, some researchers have found it useful to separate out the surface tension term and integrate it separately [35]. In practice, this amounts to simulating several small time steps for the surface tension term *alone*, within a single larger time step of the rest of the momentum equations. Because the cost of these frequent surface tension force evaluations ends up being dwarfed by the cost of a single pressure solve (the linear system necessary to enforce a divergence-free velocity field), this strategy is relatively efficient to compute. Unfortunately this *sub-cycling* strategy de-couples the surface tension term from the pressure forces in the Navier-Stokes equation, so there is no longer a guarantee that the system conserves volume. Because several individual advections due to surface tension forces can accumulate without any divergence-free constraint on their velocity fields, this strategy can easily violate Equation 16. The pressure forces and the surface tension forces do not communicate back and forth, so we can never be sure that the surface tension forces are behaving realistically with this model.

Instead of directly evaluating the surface tension forces, we can compute a temporal finite difference of mean curvature flows, as explained by Equation 22. If we sub-cycle the mean curvature flow by computing many small time steps within a single fluid simulation time step and then take the finite difference between surfaces in order to evaluate the surface tension, then we will again be left with a vector field which is not necessarily divergence-free. In general, mean curvature evolution of a closed

surface will tend to cause the surface to *lose* volume over time, giving the resulting velocity field a significantly *negative* divergence overall. However, if we use a *volume-preserving* mean curvature flow instead of standard mean curvature flow, the vector field representing the surface flow will be a finite difference between two surfaces of the same volume. Because this surface evolution maintains a constant volume, the integral of the volume flux through its surface will be zero. According to Gauss' theorem, this zero total volume flux indicates that the flow will have a divergence of zero in the continuous setting. In practice, due to the accumulation of discretization errors in our simulation and imperfections in our inter-surface mapping, we cannot guarantee that this field will be perfectly divergence-free. However, performing a volume-preserving curvature flow instead a standard mean curvature flow produces flows with a divergence very close to zero, so this new method preserves volume very well overall. Furthermore, the addition of these divergence-free surface tension velocities with the divergence-free velocity from the rest of the simulation will yield a final velocity field that is still divergence-free, so we could potentially separate the surface tension force evaluation from the pressure solve completely without worrying about violating our incompressibility constraint.

One thing to note, however, is that there are several ways to convert an arbitrary vector field into one that has zero divergence. For example, the standard pressure solve used at the end of a fluid simulation step uses a projection technique to find the closest divergence-free field in a least-squares sense. We can compute a divergence free mean curvature flow in several ways, and we have implemented two methods: we can globally re-scale the volume of each surface to guarantee that the total volume is conserved [39, 137], or we can use a more local volume preservation [43]. We have not yet tried to project the curvature flow onto the nearest divergence-free velocity field, though this idea seems promising.

8.3 Splitting High and Low Frequencies

In Section 8.1, we stated that we simulate surface tension dynamics both on the Eulerian grid and at sub-grid scales. We use our Eulerian method to simulate all surface tension forces with a low enough curvature to be resolved on the grid, and we use our mesh-based wave simulator to compute the remaining high-frequency surface tension behavior. This two-level approach helps us avoid deleting high-frequency surface features that cannot be adequately resolved on the grid, and it allows for the efficient animation of detailed surfaces.

If we want to represent the surface tension dynamics at two separate scales (a high-resolution force and a low-resolution force), then we can separate the surface tension coefficient σ into one that acts on high frequencies, σ_{hi} and one that acts on low frequencies, σ_{lo}.

$$\sigma = \sigma_{\text{hi}} + \sigma_{\text{lo}} \tag{25}$$

We can interpret our surface tension dynamics as a finite difference between mean curvature flows (Equation 22), so this split surface tension coefficient leads to two separate mean curvature flows

$$\frac{\partial C}{\partial t} = (\sigma_{\text{hi}} + \sigma_{\text{lo}})\nabla^2 C \tag{26}$$

$$\frac{\partial C}{\partial t} = \sigma_{\text{hi}}\nabla^2 C + \sigma_{\text{lo}}\nabla^2 C \tag{27}$$

We can then use standard operator splitting techniques to compute one flow at a time. For example, using first order forward differencing yields the following two equations:

$$C^* = C^{\text{old}} + \Delta t \sigma_{\text{hi}}\nabla^2 C^{\text{old}} \tag{28}$$

$$C^{\text{new}} = C^* + \Delta t \sigma_{\text{lo}}\nabla^2 C^* \tag{29}$$

where Equation 28 updates the original surface C^{old} with the high resolution flow to get a temporary surface C^*, and Equation 29 updates the temporary surface with the low resolution flow to get the final surface integrated through the entire mean

curvature flow C^{new}. Similarly, if we separately calculated surface tension forces for the high resolution flow and added them to the forces from the low resolution flow, they would sum up to the total surface tension force that would result if we performed one single curvature flow.

In our simulations, the low resolution surface tension forces will act as boundary conditions to our fully volumetric Eulerian fluid simulation, while the higher resolution tension forces will be approximated by a boundary-element based wave simulator which ignores volumetric fluid effects. Because the Eulerian surface tension forces have a better coupling with the rest of the fluid simulation, we would like them to be as accurate as possible. More specifically, we would like σ_{lo} to be as large as possible in order for a large portion of the flow to be captured on the volumetric grid. On the other hand, we cannot allow σ_{hi} to be too small, either. If C^* from equation 28 has any surface features that are too high frequency to be resolved on the fluid grid, then accuracy and stability problems may result. We therefore are faced with the problem of making σ_{lo} as large as possible while still guaranteeing that $\sigma_{\text{hi}} = \sigma - \sigma_{\text{lo}}$ is large enough to remove all high-frequency features from the intermediate surface C^*.

One solution is to make σ_{hi} so conservatively large that high frequency features never appear in the intermediate surface. This approach is simple to implement, though it is far from ideal.

Another possible solution is to adaptively choose a new σ_{hi} each fluid simulation time step such that it is as small as possible. For example, a mean curvature flow can be computed by simply choosing some small constant k and using it to compute a temporary surface $C^{n+1} = C^n + k\Delta t \nabla^2 C^n$ by causing a small amount of mean curvature flow proportional to $k\Delta t$. At the end of this small flow, we could check whether this temporary surface has a low enough curvature everywhere to be adequately resolved on the fluid grid. If so, we can use this temporary surface as C^* and

use the rest of the remaining flow for our low resolution surface tension forces. If not, we repeat this process until the temporary surface is acceptable. If we went through m surface flows before finding an acceptable surface, then $\sigma_{\text{hi}} = mk$. I believe that this adaptive selection of σ_{hi} would work well, although we did not have a chance to implement this method by the time of this work's publication [145].

Another approach to this problem exploits the fact that low-pass filtering of a geometric surface is mathematically identical to spatially varying mean curvature flow. If we low-pass filter the original surface such that the new filtered mesh has low enough curvature to be sampled on the Eulerian grid, then we can recover the amount of curvature flow experienced by each surface feature and compute a spatially-varying σ_{hi}. We could use this information to compute a new spatially-varying σ_{lo} from the constant σ. In our experiments, however, we chose to re-sample the original surface on our Eulerian fluid grid by computing its signed distance function and then extracting the zero surface with marching cubes. Unfortunately, we did not compute the amount of flow caused by this low-pass filter and instead approximated it with a constant σ_{lo} to simulate the grid based tension forces. This choice of a constant σ_{lo} introduced errors that biased our simulations to have larger surface tension forces for surface features smaller than a grid cell, though the results are still accurate for surface features larger than a grid cell. We can defend this choice by claiming that fluid forces would be inaccurate for features smaller than a grid cell anyway, because those features cannot be resolved by any other terms in Equation 15. However, we are still interested in seeking solutions to this problem that are as accurate as possible, in the event that they prove useful outside of the computer animation community.

To further elaborate on the method used in our experiments, we computed the signed distance function of the surface using the same voxelization technique discribed in Chapter 4. To ensure that even fine details of the mesh such as thin liquid sheets and small liquid drops are resolved on the grid, we triangulate the isosurface not at

zero but for a thickened level set of $\sqrt{3}/2\ \Delta x$ (thickened outward from the liquid volume). We handle each disconnected component of the mesh separately, to make sure two separate components do not erroneously merge during the reconstruction from the signed distance field. Note that despite the thickened level-set, we can guarantee that the surface after the low-pass filtering has the same volume as the input surface using the volume rescaling technique of Section 8.9.

An alternative approach to low-pass filter the mesh would be to continue collapsing all edges that are smaller than Δx. Our naive approach to this solution yielded highly temporally variable surfaces, so a better approach may be to leverage results from the mesh simplification community [62, 52]. We found it advantageous to use methods that resulted in low vertex counts, in order to cleanly interpolate forces onto the fluid grid and to make the subsequent low-resolution mean curvature flow more efficient.

Our experiments have shown that the thickened grid-based re-sampling scheme mentioned above is faster and more consistent than the purely Lagrangian approach based on edge collapses. The grid-based scheme prohibits topological artifacts due to thin regions, although it can cause close regions of a single component to merge together. We did not notice any artifacts in practice, so we used this scheme to generate the examples of Section 8.11.

8.4 Mean Curvature Flow

Sussman et al. [137] showed that they can compute surface tension forces for a level-set based implicit surface by taking the finite difference between mean curvature flows on the Eulerian grid. In order to better couple the pressure forces to the surface tension, they constrained their mean curvature flow such that it preserved volume. They chose to use global volume preservation by approximately conserving the total volume per fluid component, by subtracting the average curvature from the curvature

evaluated at each point on the surface:

$$\frac{\partial C}{\partial t} = \sigma(\nabla^2 C - (\nabla^2 C)_{\text{avg}}). \tag{30}$$

Note that this global curvature averaging approximates total volume conservation per fluid component, but it will produce unphysical effects for surfaces with highly variable curvatures. For example, if we wish to simulate the surface of a calm pond with a small splash in the center, the curvature averaged over this extremely large plane will be negligibly small. If we subtract this near-zero curvature from the curvature at the splash location, it has an insignificant affect, and we might as well be simulating a normal mean curvature flow without subtracting any averaging. As a result, the small splash will not experience any local volume preservation effects, and any small tendrils will shrink and disappear instead of experiencing more physically correct bulging and droplet pinchoff behavior. For this reason, a mean curvature flow with a more local volume preservation is desired.

Our method uses a mesh-based surface instead of an implicit level-set surface, so we have to make some significant changes to the approach. Instead of running a simple diffusion equation on the signed distance function of a level-set surface, we explicitly compute the mean curvature flow on the surface itself. We discretize the mean curvature flow with the standard Laplace-Beltrami operator according to [117, 39]. For a function f that takes the value f_i at a vertex v_i, the discrete Laplace-Beltrami operator can be formulated as

$$\nabla^2 f = \frac{1}{2A_i} \sum_{v_k \in \mathcal{N}(v_i)} (\cot \alpha_k + \cot \beta_k)(f_k - f_i) \ , \tag{31}$$

where the area of the barycentric dual cell associated with a vertex v_i is denoted by A_i, and the neighborhood $\mathcal{N}(v)$ of a vertex v consists of the adjacent vertices v_k. To compute the curvature at each point, we use the position of each surface vertex $\mathbf{x}_i$ for the function f_i.

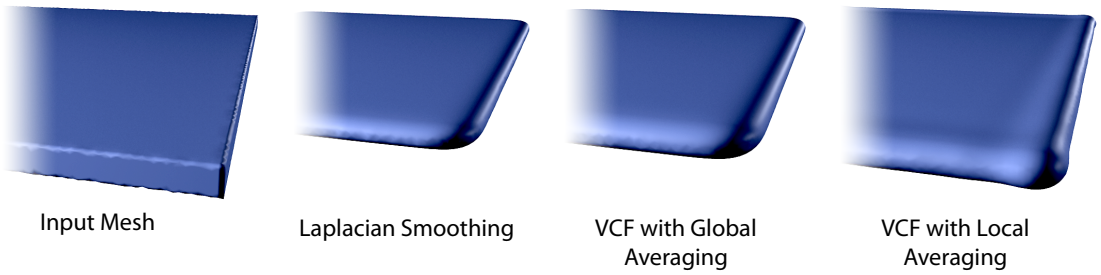


Figure 50: A comparison of the different techniques for curvature flow: The input mesh is shown on the left, and Laplacian smoothing is shown second. Note that volume-preserving mean curvature flow with global averaging (third) behaves similarly to Laplacian smoothing, but it conserves volume by pushing the entire surface outward. In contrast, volume-preserving mean curvature flow with local averaging [43] (right) conserves volume by locally bulging the surface where the most significant flow has occurred.

This general form can be utilized to create a volume-preserving surface flow by subtracting the average curvature *per surface component* from the curvature evaluation at each vertex, analogous to the volume-preserving level-set flows of Sussman et al. [137] (Equation 30). Alternatively, we can instead average the surface curvature *locally within each vertex neighborhood* following Eckstein et al. [43] in order to produce more physically plausible surface tension effects. In Figure 50, we illustrate the effects of each of these curvature flows.

In our experiments, we explicitly integrated all of our surface tension flows over many small time steps within a single fluid time step. Explicit integration of these curvature flows is fairly expensive, because the maximum time step size is proportional to the length of the surface's smallest edge length squared. Explicit integration allowed us to actively update the surface via edge collapses or subdivisions for extraordinarily large curvature flows, however. A more efficient alternative would be to use an implicit integration of the mean curvature flows, as described by [43]. However, very large flows can lead to convergence problems, for example, in the presence of thin features caused by a Rayleigh-Plateau instability. A faster, but less accurate alternative to computing this type of volume-preserving curvature flow is to use a

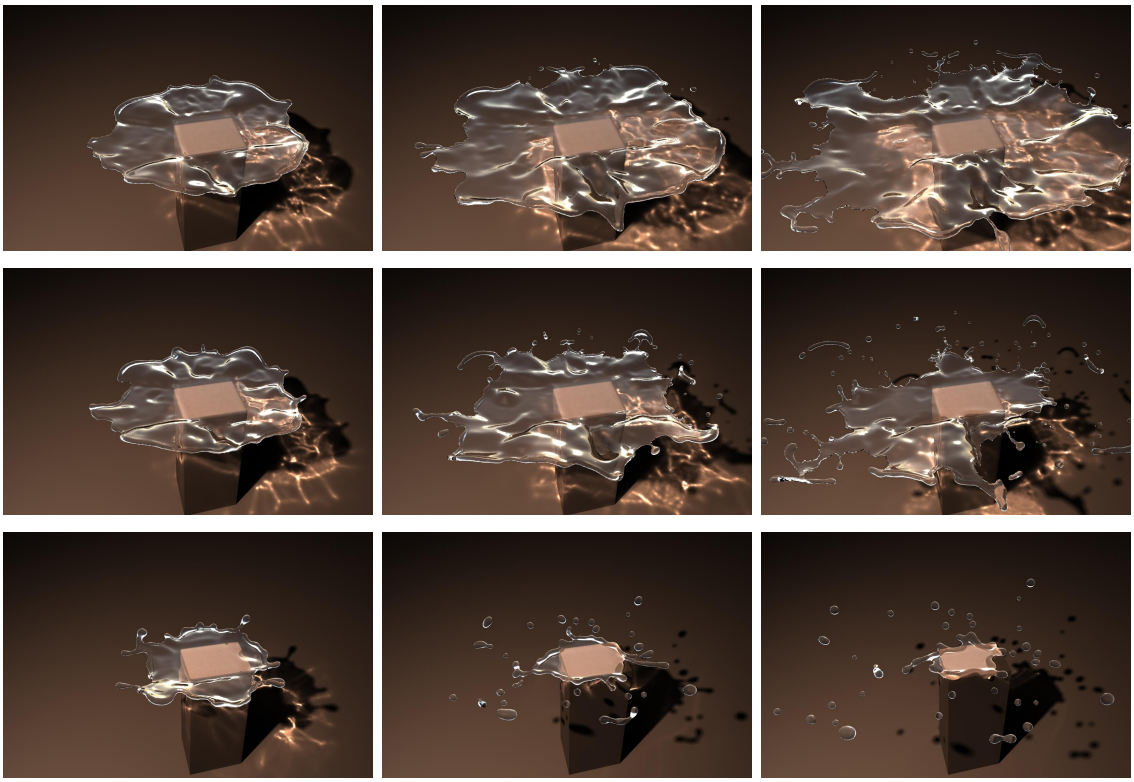


Figure 51: A fast-moving drop of water collides with a planar obstacle, resulting in a horizontal sheet of liquid. These images show simulations with increasing surface tension (from top to bottom) at the same instants in time. Stronger surface tension causes the liquid sheet to break up earlier.

global average curvature, as in Equation Equation (30) by computing a single volume-weighted average curvature for each component. We used both the local and global volume-preserving mean curvature flow methods interchangeably for examples in this chapter. While Figure 51 and Figure 46 make use of the local curvature averaging, the other examples below make use of the faster per component averaging.

In a next step, we use the mesh resulting from this curvature flow to solve for the surface dynamics on the input mesh F. The following section will describe how to compute correspondences between the two meshes, which will be used to initialize the wave equation in Section 8.6 and set the boundary conditions on the grid in Section 8.8.

8.5 Mesh Correspondence

So far we have described how to take the input surface mesh F and run a low-pass filter on it to compute a smoother mesh S whose curvatures can be adequately resolved on the Eulerian grid. We then described how to run a volume-preserving mean curvature flow to transform S into an even smoother mesh T.

In order for us to simulate surface tension waves on the sub-grid cell level, we need the input surface F to oscillate about it's smoother counterpart S. The simulation of these oscillations requires us to map the vertices of the original mesh F to their closest points on S. Additionally, in order to compute a finite difference of mean curvature flows and use the resulting surface tension forces in the Eulerian fluid simulation, we need to map the vertices of S to their closest points on T.

To compute these point-to-point correspondences efficiently, we can use any spatial data structure, such as grids, trees or hash tables. For our implementation, to compute correspondences between F and S, we have chosen to insert the locations of the vertices $\mathbf{x}^S$ of S into a kd-tree and query the closest point $\tilde{\mathbf{x}}_i^S$ to each vertex $\mathbf{x}_i$ of F using this tree. We then check whether there is an even closer point $\mathbf{x}_i^S$ to $\mathbf{x}_i$ on the triangle fan around $\tilde{\mathbf{x}}_i^S$. Note that, due to fast motions in our simulations, the surface might have shifted, and we have to make sure not to compute the closest point on the wrong side of thin fluid surfaces. To prevent this surface inversion, we only return points from the kd-tree query whose normals face the same direction as the normal of $\mathbf{x}_i$ (that is, the dot product between both normals must be positive). After this step, each vertex $\mathbf{x}_i$ on F has a corresponding point $\mathbf{x}_i^S$ on S. The set of points $\mathbf{x}_i^S$ on S will later on represent the reference surface with respect to which we solve the wave equation. The signed distance h_i between the two corresponding points can be computed with

$$h_i = (\mathbf{x}_i^S - \mathbf{x}_i) \cdot \mathbf{n}_i^S \ , \tag{32}$$

where $\mathbf{n}_i^S$ is the normal of the surface S at $\mathbf{x}_i^S$. These distances can now be used to solve for the surface dynamics.

8.5.1 Mesh Correspondence as a Homeomorphism

One interesting thing to note is that the correspondence computation in this section is identical to explicitly computing a *homeomorphism* between two meshes [44]. This observation implies that this correspondence problem will fail if the surfaces are not *homeomorphic*. For this reason, we must take great care to ensure that our transformations from F to S and from S to T do not change topology in any way.

We explained how we took extra care to make sure that our low-pass filtering tactic in Section 8.3 was able to resolve very thin features, in order to avoid changing topology when filtering the mesh. Furthermore, we can guarantee that we do not change topology when computing our mean curvature flow from S to T, because the numerical integration is simply a deformation of surface vertices with some topology-preserving edge collapses [41].

Conversely, such a guarantee is impossible with level-set based mean curvature flow integration. As a result, the methods of Sussman and Ohta [137, 138] provide no guarantee the existence of a mapping between two surfaces when computing surface tension forces. Thin surface features are especially vulnerable to topological change during mean curvature flows, and this problem cannot be solved with increased grid resolution. As discussed in Chapter 7, Rayleigh-Plateau instabilities are a natural mechanism for surfaces to become infinitely thin in a finite amount of time. Similarly, volume-preserving mean curvature flows can also produce extremely thin structures in the same scenarios. The correct behavior of our simulation should be to find a mapping between the original surface and this incredibly thin structure, and produce surface tension forces driving the surface thinner and thinner, until it is thin enough to reach some topological cut-off. A level-set-based curvature flow, however, will

delete the thin feature completely, causing the surface mapping algorithm to find its closest point on a surface much farther away. As a result, these surface tension forces on level set methods will not even be close to the correct *direction*, let alone having an accurate magnitude. Consequently, the surface tension forces when using a finite difference of mean curvature flows on a grid will reliably become wildly inaccurate or unstable in the presence of thin features. Again, these problems with level-set based curvature flows are caused when no homeomorphism exists between the two surfaces. Consequently, the use of methods for forbidding topology changes in the level set [123, 15] may eliminate the problem altogether.

8.6 Wave Simulation on the Mesh

To efficiently compute the motion of capillary waves on the mesh surface, we linearize the surface tension dynamics with the classical wave equation. The wave equation is an often-used second order differential equation in which the normal acceleration of vertical elevations h is related to their second derivatives:

$$\frac{\partial^2 h}{\partial t^2} = c^2 \nabla^2 h \ . \tag{33}$$

The values of h for each vertex of F are given by Equation (32), and $c^2 = \sigma_{\mathrm{hi}}$.

We perform the wave simulation on the triangle mesh F that represents the liquid surface. Similar to [4], we use a finite element discretization of the height values. The surface height is a scalar function that is represented with linear basis functions, and whose values h_i are located at the vertices of the triangle mesh. This leads to the commonly used operator with cotangent weights for the second derivatives according to Equation (31).

To overcome the problem of energy dissipation while ensuring robustness, we make use of the Implicit Newmark time integration scheme [107] that was proposed in [4]. In the following, we will denote the vector of the height values for all vertices with $\mathbf{h}$ and write the evaluation of the Laplacian of $\mathbf{h}$ as the multiplication with the matrix

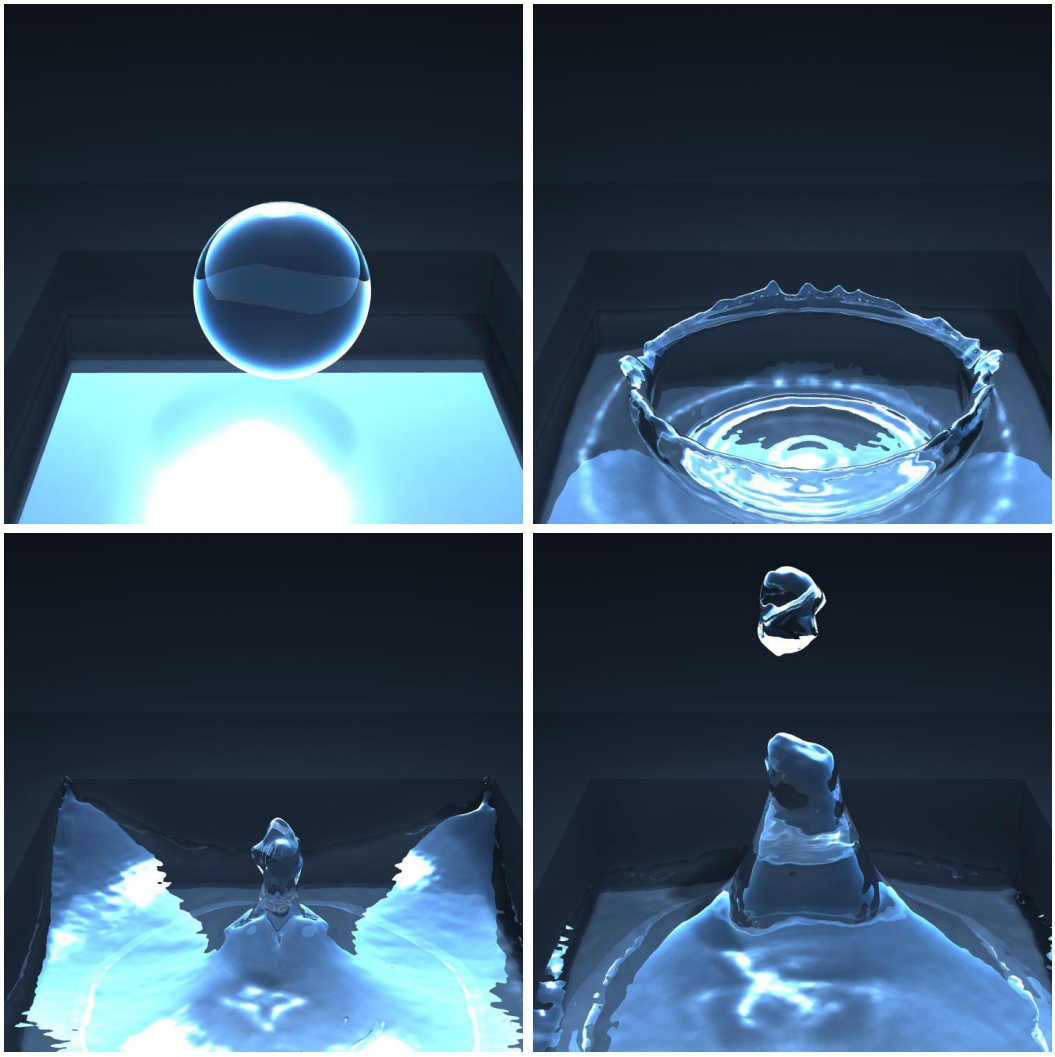


Figure 52: A drop falls into a body of water. The splash creates a Worthington jet, and the strong surface tension forces create a Rayleigh Plateau instability, resulting in the column breaking apart as a droplet pinches off.

L. Now, given a time step Δt, the Newmark integration step in its unconditionally stable form is given by

$$\left(I - \frac{\Delta t^2}{4} c^2 L\right)\mathbf{h}' = \mathbf{h} + \Delta t \frac{\partial \mathbf{h}}{\partial t} + \frac{\Delta t^2}{4} \frac{\partial^2 \mathbf{h}}{\partial t^2}. \tag{34}$$

These equations represent a sparse system of linear equations; the unknowns are the positions of the next time step $\mathbf{h}'$ that can be solved with a standard iterative solver such as a conjugate gradient method.

As the Newmark integrator is a symplectic scheme, this leads to a conservation

of energy when solving the wave equation. We can manually introduce viscosity into the wave equation solve by manually scaling the surface heights by a factor of slightly less than one relative to the average height of the surface. After solving the wave equation, we have a new height h_i' for each of surface F's vertices. The updated positions of the vertices is computed with

$$\mathbf{x}_i = \mathbf{x}_i^S + h_i' \mathbf{n}_i^S \ . \tag{35}$$

8.7 Steady-State Surface Tension Approximation on the Mesh

While the method for simulating dynamic surface tension proposed in Section 8.6 produces visually-appealing ripple effects on the surface, we can also efficiently simulate damped surface tension effects by neglecting surface wave phenomena entirely. As mentioned previously, we can use volume-preserving mean curvature flow to produce a reference surface S from the original surface mesh F. The sub-grid scale capillary waves oscillate about this surface, and they will eventually converge to it after the wave motion damps out. Therefore, S represents a type of steady-state solution to the surface tension dynamics. To efficiently simulate surface tension in the absence of small-scale capillary waves, all we have to do is run volume-preserving mean curvature flow on the original surface F by an amount proportional to σ_{hi} at each time step of the simulation. Because this approximation can be efficiently computed even on high resolution surface meshes, it can be used to simulate detailed droplet effects (See Figures 51 and 46).

8.8 Eulerian Surface Tension Forces

In Sections 8.6 and 8.7, we discussed how to simulate surface tension phenomena on the surface mesh for features smaller than a grid cell. In this section, we will discuss the steps necessary for simulating surface tension on the fluid simulation grid, in order to capture lower resolution, larger-scale capillary effects.

As in [137], we approximate the surface tension forces with a finite difference of mean curvature flows. We use the surface correspondences computed in Section 8.5 to find the distance between vertices on our low-pass filtered mesh S and points on our smoothest mesh T. We take these distances and divide them by the time step size Δt in order to approximate surface tension forces at each point on S, as discussed in Section 8.2.

Next, we need to transfer these surface tension forces from the mesh S to all of the grid cells near the liquid-gas interface. As the points in S will typically not align with the centers of the grid cells, we again use a kd-tree to retrieve the closest point on S with respect to the cell center. We interpolate the data if a closer point is found on an adjacent triangle. For this step, we can estimate the normal at the cell from the signed distance field of S and only query points from the kd-tree with aligning normals.

Note that we used one of the simplest possible methods for transferring data from the mesh to the grid; we simply found the closest point on the mesh and copied it over. Many more accurate methods exist for transferring data from a surface to a volume, such as the "immersed boundary methods" of Peskin [116, 115]. Though we did not implement any of these more accurate data transfer schemes, we would like to investigate similar methods in the future. For the purposes of the animations in this chapter, the simplest method sufficed.

After using this surface tension value as a boundary condition for the Eulerian fluid solver, we then proceed to advect the surface mesh F according to the fluid velocities and handle topological changes with either of the methods from Chapter 6 [154] or Chapter 7 [155].

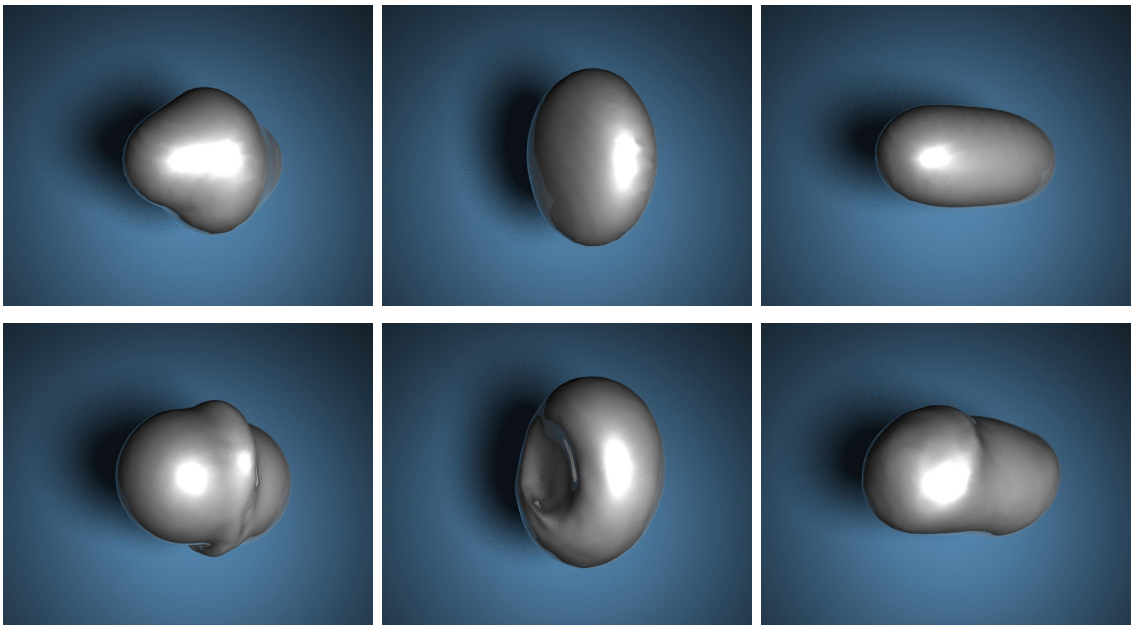


Figure 53: Comparison of a level set based surface tension forces (top row) with our method (bottom row). While both simulations show an overall similar behavior of the two merging drops, our method is able to capture a capillary wave travelling around the merged drop with a high speed. In addition, our method is much better at conserving volume.

8.9 Mass Conservation

Despite the accurate fourth-order advection of our mesh vertices, the overall conservation of mass cannot be guaranteed due to the accumulation of several smaller errors throughout the fluid simulation time step (tri-linear interpolation of velocities at the surface vertices violates the divergence-free condition, for example). In addition, steps such as surface subdivision, topological changes, or the re-initialization from the wave equation solve can cause additional violations of the conservation of mass. This is especially noticeable in smaller liquid volumes, which could completely disappear due to the comparatively increased significance of these small errors. This section will explain how we can take advantage of our triangle mesh surface representation in order to correct for the effect of these errors and explicitly conserve mass throughout our fluid simulation.

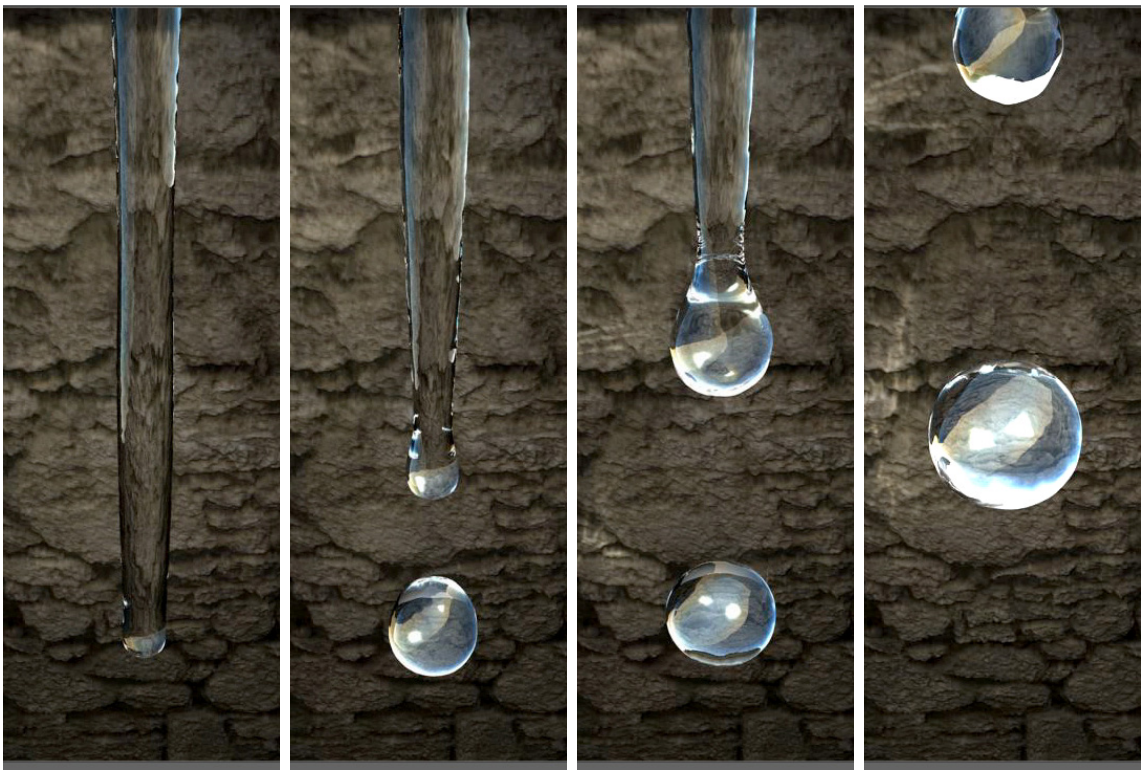


Figure 54: Example of the self-reinforcing instability of a fluid jet causing droplet pinch off. The images from left to right show the change in behavior when increasing the surface tension of the liquid (σ_s ranges from 0.00003 to 0.01). Table 1 lists the simulation data for the second simulation from the left.

Fortunately, because our surface representation is a triangle mesh, we can accurately and efficiently compute its volume at any time [83]. Just like Müller [98], we iteratively rescale our triangle mesh along the surface normals in order to correct for changes in mass at the end of each time step. Given a desired volume V_{desired} and a current volume V_{current} with surface area A_{current}, we choose a step size of

$$\Delta t_{\text{mass}} = \frac{V_{\text{desired}} - V_{\text{current}}}{A_{\text{current}}} \tag{36}$$

and we iterate the following steps. First, we update the positions of all surface vertices with $\mathbf{x}_i = \Delta t_{\text{mass}} \mathbf{n}_i$, where $\mathbf{n}_i$ is the normal of vertex i. We then recompute V_{current}. If the relative error $\epsilon = (V_{\text{desired}} - V_{\text{current}})/V_{\text{desired}}$ changes sign from one iteration to the next, we reinitialize the step size with $\Delta t'_{\text{mass}} = -\Delta t_{\text{mass}}/2$. This process is repeated until converging to the desired accuracy. We have used $|\epsilon| < 0.005$ for the examples

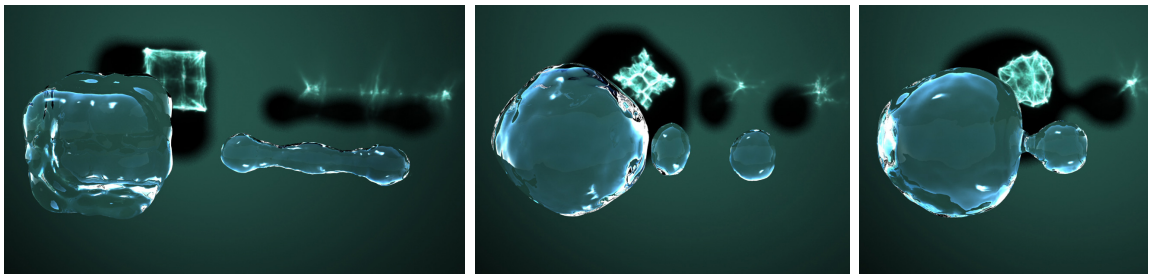


Figure 55: These images show a simulation purely with our sub-grid wave equation solver, and without any Eulerian surface tension forces. While the right drop is moving towards the large drop, it splits up and finally merges with the large drop. Throughout the simulation, detailed capillary waves can be seen on the surfaces of the drops.

in this chapter. Usually one or two iterations sufficed to reach our desired accuracy.

Note that similar to the other steps of our algorithm, it is important to perform the volumetric rescaling separately for each connected component of the surface. Components with a negative volume, like bubbles, can be treated in the same way. Using either of the algorithms described in Chapter 6 [154] or Chapter 7 [155] to handle topology changes of the surface, we only need to identify disconnected components and update the target volume V_{desired} for each component when a topology change is registered. These topological changes happen infrequently compared to the overall number of simulations steps.

8.10 Input Parameters

The algorithm presented in this chapter depends on a number of parameters in addition to the parameters required by the Eulerian fluid simulation. This method requires the user to set a grid-based surface tension strength σ_{lo}, a mesh-based wave speed σ_{hi}, and a time step size for the volume preserving mean curvature flows. If we choose to use the steady state surface tension approximation in Section 8.7, we must also choose an appropriate smoothing strength and time step size.

Although relatively few parameters must be chosen for the surface tension algorithm, its implementation still requires a relatively delicate combination of different

simulation items. As in previous chapters, we must take care to maintain the triangle surface mesh. However, because curvatures are being computed directly from this mesh, we must take care to strictly enforce a minimum edge length and put bounds on the minimum and maximum allowed angles in order to ensure a stable simulation.

When we re-sample the triangle mesh due to topological changes, subdivisions, or edge collapses, we also re-sample any data stored on the mesh. This re-sampling strategy significantly affects the surface wave physics, and our current strategy of naively averaging nearby data leads to significant damping of surface waves. A careless re-sampling strategy here can inject energy into the system, leading to eventual numerical instabilities.

The performance of the algorithm in this chapter is also closely tied to the underlying fluid simulation parameters. For example, the grid used for topological changes determines the minimum feature size allowed in the grid-based simulation. Figure 54 uses a grid that is only 15 grid cells across, so we cannot expect any high-resolution satellite droplets to be generated during each pinch off.

In addition, the resolution of the fluid simulation grid size directly affects the momentum of the fluid. This effect is especially noticeable when using semi-Lagrangian advection. A low resolution grid tends to inject artificial viscosity into the simulation, which can cause undesirable damping in a highly energetic surface tension simulation.

8.11 Results

The following simulations and performance measurements were performed on a standard PC running at 3GHz without multi-threading. First, we compare our method to a surface tension computation performed by calculating the curvature of a level set surface representation. Several frames of a simulation of two merging drops with strong surface tension can be seen in Figure 53, where the top row was performed with the level-set, and the bottom row with our method. In each simulation, the

drops have an initial diameter of 9 cells. While both simulations result in a similar behavior, our method is able to capture the waves on the surface of the drop that are caused by the merge. In addition, our method preserves the volume of the fluid and does not start to drift. However, the level-set style simulation took less time to compute. We also compared the stability of both methods with this setup: while level-set based simulation quickly becomes unstable with an increase in surface tension, our approach gives violently energetic, but reasonable, results even after increasing the surface tension coefficient by a factor of fifty. Because the time step restriction increases super-linearly for such explicit schemes, this difference in stability increases strongly at higher simulation resolutions. Consequently, the relative stability of our method (Equation 24) will only improve with more detailed simulations.

The simulations shown in Figure 54 highlights that our method is able to efficiently resolve droplet pinch off for liquid jets. This form of instability is caused by slight perturbations of the initial cylinder that grow over time. If the diameter of the jet is small enough, this ultimately leads to a pinch off. This phenomena is known as the *Rayleigh-Plateau* instability, and has been widely studied in experiments and simulations. See [26] and Section 7.2.1 for details. For our simulation, the jet is resolved with less than 5 grid cells in diameter, and the simulation ran at five frames per second. The images from left to right show how the behavior of the jets differs when increasing surface tension. The leftmost image has a surface tension coefficient close to zero, resulting in barely any droplet pinch off, while the right-most one shows a simulation with strong surface tension, causing drops to pinch off right below the inlet.

An example of a larger fluid simulation with a resolution of 128^3 can be seen in Figure 52. Here, a larger drop is falling into a body of water, and the resulting back splash causes a single drop to pinch off at its end. In this case, the high fluid simulation resolution is necessary to counter the numerical viscosity of the fluid simulation's

semi-Lagrangian advection scheme.

The non-oscillatory surface tension approximation of Section 8.7 is demonstrated in Figure 51, where no grid-based surface tension was active. The setup is a well studied experimental one: a liquid drop (or jet) with high velocity impacts a planar or curved obstacle and forms a horizontal liquid sheet [32]. The sheet breaks up at a specific radius inversely proportional to the surface tension strength. Our method allows us to re-create the effect that high surface tension causes an early break up, while low surface tension results in a large spreading sheet. For this simulation, we make use of the technique presented in Chapter 7 [155] to track the very thin liquid sheets.

Finally, a simulation of a crown splash is shown in Figure 46. This phenomenon develops when a drop hits a shallow volume of water, and the evolving circular liquid sheet breaks up into droplets at its outer boundary. For this simulation, we have used a combination of the grid-based surface tension forces and the non-oscillatory approximation. By varying the strength of the grid-based component, we are able to control the evolution of the liquid sheet, as seen in the accompanying video. To our knowledge, we are the first in the computer graphics and computational science field to simulate a full crown splash with droplet pinch off, and our method allows us to efficiently perform a very detailed simulation with a relatively low grid and surface resolutions.

8.12 Discussion

The complexity of our algorithm predominantly depends on the resolution of the surface discretization. For a liquid surface mesh F with n nodes and a smoothed version S with $m < n$ nodes, we observe that the following steps have a complexity of $O(n)$: the signed distance field computation, the calculation of correspondences from F to S, and solving the wave equation on F. On the other hand, the following steps

Table 1: This table shows the settings used for our simulations. The mesh resolution is given relative to the grid size, while the timing is the average simulation time per frame of animation. σs with an asterisk * denotes simulations with the non-oscillatory approximation.

	grid	mesh	time	σ_g	σ_s
Figure 53	75^3	0.5	1.0s	0.4	0.015
Figure 54	$15^2 \times 75$	0.5	0.2s	5e-5	4e-3
Figure 51	$160^2 \times 80$	0.25	12.6s	0	1.5e-4*
Figure 52	128^3	0.5	14.1s	2e-4	2e-4
Figure 55	50^3	0.45	1.0s	0	2.25e-3
Figure 46	$180^2 \times 50$	0.55	22.3s	3e-6	3.5e-4*

have a complexity of $O(m)$: computing the volume preserving mean curvature flow for S and T, the correspondences from S to T, and the calculations of the boundary conditions for the grid-based simulation. The magnitude of both n and m depends on the surface area of the liquid interface and the average triangle size, instead of the grid resolution.

The simulation settings and computation times for the shown simulations can be seen in Table 1. The fastest simulation was the Rayleigh-Plateau instability from Figure 54 with 0.2 seconds per timestep on average, while larger simulations such as the crown splash from Figure 46 require up to 22.3 seconds per timestep. For all of the simulations shown, we perform a single fluid simulation time step per frame of animation. The resolution of the fluid simulations is typically determined by how many cells the solver requires to achieve a viscosity that is low enough to produce the desired motion. Our surface meshes are typically parametrized relative to the grid resolution, and the small simulations such as Figure 54 have around 2100 vertices for the mesh, while larger ones such as Figure 46 resolve the surface with up to 270,000 vertices. Similarly, the amount of time required by our algorithm is determined by the complexity of the surface. For larger fluid simulations, such as Figure 52, on average one third of the overall time is spent for the surface tension calculations, while others, such as the large surfaces of Figure 51, require two thirds of the overall

time to compute the surface tension dynamics.

Note that we model the non-linear surface tension behavior of a fluid simulation with a linear wave equation. We originally implemented a fully non-linear wave solution for the sub-grid scale dynamics, but our early attempts were plagued with self-intersections and numerical instabilities. The linearized version avoids these problems. However, the linearization also means that we can only simulate a single wave propagation speed given by the choice of the parameter c in Equation (33). However, we achieve a non-linear behavior even without an underlying fluid simulation due to the fact that the positions updated by the wave equation solve are the input for the next mesh simplification and smoothing step, as described in Section 8.4

An important property of the wave equation is its ability to locally preserve the volume of the surface height function. This means that it is able to capture, for example, the effect of a bulging front of a fluid sheet even when the sheet is much smaller than a grid cell. It can thus capture effects such as the breakup of thin sheets and droplet pinch off without having to rely on a Eulerian fluid simulation. However, because the wave equation dynamics in Section 8.6 oscillate about a down-sampled surface while the non-oscillatory approximation in Section 8.7 use a genuine volume-preserving mean curvature flow to compute the final shapes, the non-oscillatory approximation yields more accurate bulges and droplet pinches for features significantly smaller than the grid resolution.

Our approach to surface tension does have some limitations. The surface wave equation does not completely conserve the mass of the overall fluid, but more specifically, that of the represented heights. This fact, in combination with the inaccuracies of the iterative solve make it necessary to enforce mass conservation with the method explained in Section 8.9. Conveniently, we can accurately measure the initial volume of each component and keep its mass constant. In addition, despite the energy-conserving nature of our Newmark solver, the wave equation can lose energy due to

the re-sampling of the underlying mesh. We currently only linearly interpolate the wave equation variables for triangle subdivisions and edge collapses, so higher-order interpolations could help to conserve energy.

The simulations in Figures 46 and 51 generate particularly thin sheets of liquid. These thin sheets can cause problems for the surface wave simulation in Section 8.6 when the simulation creates waves with amplitudes larger than the thickness of the sheet. In this case, the surface waves can intersect the other side of the liquid sheet and prematurely trigger a topological change. The non-oscillatory surface tension simulation works well for these examples, because it does not overshoot and cause unintended self-intersections.

Lastly, as with any grid based method, our Eulerian approach for surface tension forces can be inaccurate for fluid components the size of a grid cell, because the boundary conditions can not be accurately represented on this scale anymore. We can, however, reduce the strength of the Eulerian surface tension for connected components with a volume on the order of a grid cell, because we can rely on our sub-grid model to handle its dynamics.

8.13 Conclusion and Future Work

We have presented a method to simulate surface tension flows using a mesh-based surface representation. Our method handles grid based surface tension forces with high stability and allows for efficient simulations of fast capillary waves on the liquid surface. This enables detailed simulations featuring strong surface tension forces with droplet pinch off as well as capillary waves. In addition, we have shown how to use a fast approximation of the wave equation solution with a steady state surface flow to yield highly detailed surfaces. The mesh surface makes it possible to accurately compute volume preserving mean curvature flow, which is the basis for our grid based forces and sub-grid dynamics. Our approach allows us to simulate a wide range of

surface tension phenomena with a low computational cost.

There are a number of extensions to our basic technique that we are considering as future work. We would like to include a non-linear wave solver in order to handle dispersive capillary waves on the liquid surface more accurately. In addition, our method could be used to create interesting effects for bubbles and underwater phenomena, instead of only drops. For the grid based surface tension component, we would like to use more accurate immersed boundary methods from [115]. Finally, it would be intereting to include interactions with obstacle surfaces by enforcing contact angles [151, 138].

CHAPTER IX

CONCLUSION

Within this dissertation, I have presented methods for simulating various physical systems in which highly detailed surface phenomena plays an important role. Using these techniques, the bulk of the physics can be simulated with either Lagrangian finite element or Eulerian finite difference methods. Although several methods exist for simulating the interface between different physical media, this document has focused exclusively on the use of an explicit Lagrangian triangle mesh for the interface representation. The explicit Lagrangian nature of this surface ensures that any re-sampling of the surface will be kept to a minimum, allowing vastly more detailed physical surfaces than were previously possible in the computer animation literature.

Chapter 3 described a method for simulating viscoelastic fluids and solids with a finite element discretization of the physical dynamics and an embedded triangle mesh for the visible surface and contact calculations. The combination of a frequently remeshing finite element simulation with an embedded surface mesh yields much more detail in a viscoelastic fluid simulation than was possible before. At the same time, the de-coupled resolutions of the highly detailed surface and the minimally-sampled physical system allows for rapid simulation speeds without sacrificing visual quality.

Chapter 4 showed how to remove the explicit triangle mesh surface from a finite element simulation and insert it into an Eulerian fluid simulation. Using many of the same principles from Chapter 3, we can use a Lagrangian triangle mesh to simulate detailed liquid behaviors at rapid speeds, even at low computation resolutions.

Although the methods described in Chapters 3 and 4 provide compelling example simulations with highly detailed surfaces, these surfaces lacked the ability to change

topology. This inability to change surface topology robustly and efficiently throughout a physical simulation creates a severe technical limitation, significantly constraining the scenarios in which these methods can be useful in practice. Chapter 5 explained this problem in more detail, and Chapter 6 provided a simple and robust method for overcoming this limitation and allowing topological changes to a deforming mesh. In particular, adding the ability to handle topological changes to the simulation methods presented in Chapters 3 and 4 allowed them to simulate a much larger class of more deformable, more energetic, and more interesting physical scenarios.

The results from Chapter 6 show that the idea of augmenting a Lagrangian mesh with the ability to handle topological changes shows a large amount of potential, and we pursued this idea further in Chapter 7 in order to ensure that the style of topological changes in our liquid simulations was much closer to the way that liquid surfaces change topology in nature. As a result of these physically based topology changes, our simulations are now able to produce arbitrarily thin geometric features without being artificially destroyed due to an under-resolved computational method or an unphysical algorithm for numerically integrating the underlying partial differential equations.

The method described in Chapter 7 allowed us to realistically change surface topology while retaining important surface details. As a direct consequence of this work, we were able to create the new method for simulating surface tension described in Chapter 8. This new method utilized the Lagrangian nature of our surface mesh in order to create detailed ripples on top of fluid surfaces, and it allowed for a different discretization of the surface tension term in the Navier-Stokes equations that is simultaneously more detailed and more stable than previous methods in the computer animation or computational physics communities. In particular, due to the de-coupled resolutions of the surface physics and the volumetric liquid physics, we were able to simulate a full crown splash with droplets pinching off at very efficient

computational speeds.

The idea of using a Lagrangian surface mesh for computational interfaces is quite useful in computational animation and physics applications, because it easily retains large amounts of detail without excessive numerical re-sampling. The main weakness of these methods in the past was their inability to handle topological changes. This dissertation introduces new techniques for quickly and robustly handling topological changes between deformable surface meshes, overcoming the main limitation in this area of computer simulation.

After breaching the main barrier to progress in these Lagrangian mesh-based methods, we showed that there are several avenues for new work with significant impact to the relevant research communities. Notably, this dissertation also introduces a method for allowing physics-based topology changes with arbitrarily thin geometric features and a method for simulating dynamic surface tension effects with minimal instabilities or artificial damping. We have just begun to see the potential of these Lagrangian mesh-based methods with the new ability to robustly handle topological changes, and we expect the techniques discussed in this dissertation to inspire a large amount of new research into similar physical simulation methods in the near future.

REFERENCES

[1] Adalsteinsson, D. and Sethian, J., "A fast level set method for propagating interfaces," *J. Comp. Phys.*, vol. 118, pp. 269–277, 1995.

[2] Adams, B., Pauly, M., Keiser, R., and Guibas, L. J., "Adaptively sampled particle fluids," *ACM Trans. Graph.*, vol. 26, no. 3, p. 48, 2007.

[3] Alliez, P., Cohen-Steiner, D., Yvinec, M., and Desbrun, M., "Variational tetrahedral meshing," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 617–625, 2005.

[4] Angst, R., Thürey, N., Botsch, M., and Gross, M., "Robust and Efficient Wave Simulations on Deforming Meshes," *Computer Graphics Forum*, vol. 27 (7), pp. 6, 1895 – 1900, October 2008.

[5] Barber, C. and Huhdanpaa, H., "Qhull Software Package," 1995.

[6] Bargteil, A. W., Goktekin, T. G., O'Brien, J. F., and Strain, J. A., "A semi-Lagrangian contouring method for fluid simulation," *ACM Trans. Graph.*, vol. 25, no. 1, pp. 19–38, 2006.

[7] Bargteil, A. W., Wojtan, C., Hodgins, J. K., and Turk, G., "A finite element method for animating large viscoplastic flow," *ACM Trans. Graph.*, vol. 26, no. 3, p. 16, 2007.

[8] Batty, C., Xenos, S., and Houston, B., "Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids," in *Computer Graphics Forum*, vol. 29, pp. 695–704, Wiley Online Library, 2010.

[9] Batty, C., Bertails, F., and Bridson, R., "A fast variational framework for accurate solid-fluid coupling," *ACM Trans. Graph.*, vol. 26, no. 3, p. 100, 2007.

[10] Batty, C., Xenos, S., and Houston, B., "Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids," in *Proceedings of Eurographics*, 2010.

[11] Bernstein, G. and Fussell, D., "Fast, exact, linear booleans," in *Proceedings of the Symposium on Geometry Processing*, pp. 1269–1278, Eurographics Association, 2009.

[12] Bhaniramka, P., Wenger, R., and Crawfis, R., "Isosurfacing in higher dimensions," in *Proceedings of the conference on Visualization'00*, pp. 267–273, IEEE Computer Society Press Los Alamitos, CA, USA, 2000.

[13] Bhaniramka, P., Wenger, R., Crawfis, R., and others, "Isosurface construction in any dimension using convex hulls," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 2, pp. 130–141, 2004.

[14] Bischoff, S. and Kobbelt, L., "Structure preserving CAD model repair," *Computer Graphics Forum*, vol. 24, no. 3, pp. 527–536, 2005.

[15] Bischoff, S. and Kobbelt, L., "Sub-voxel topology control for level-set surfaces," *Comput. Graph. Forum*, vol. 22(3), pp. 273–280, 2003.

[16] Botsch, M. and Kobbelt, L., "An intuitive framework for real-time freeform modeling," in *ACM SIGGRAPH 2004 Papers*, p. 634, ACM, 2004.

[17] Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., and Levy, B., *Polygon Mesh Processing*. AK Peters, 2010.

[18] Botsch, M., Pauly, M., Wicke, M., and Gross, M., "Adaptive space deformations based on rigid cells," *Computer Graphics Forum*, vol. 26, no. 3, pp. 339–347, 2007.

[19] Bredno, J., Lehmann, T. M., and Spitzer, K., "A general discrete contour model in two, three, and four dimensions for topology-adaptive multichannel segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 5, pp. 550–563, 2003.

[20] Bridson, R., Marino, S., and Fedkiw, R., "Simulation of clothing with folds and wrinkles," in *Proc. Symposium on Computer Animation*, pp. 28–36, 2003.

[21] Bridson, R., *Fluid Simulation for Computer Graphics*. A K Peters, 2008.

[22] Bridson, R., Fedkiw, R., and Anderson, J., "Robust treatment of collisions, contact and friction for cloth animation," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 594–603, 2002.

[23] Brochu, T. and Bridson, R., "Robust topological operations for dynamic explicit surfaces," *SIAM Journal on Scientific Computing*, vol. 31, no. 4, pp. 2472–2493, 2009.

[24] Brochu, T., "Fluid animation with explicit surface meshes and boundary-only dynamics," Master's thesis, University of British Columbia, 2006.

[25] Brochu, T., Batty, C., and Bridson, R., "Matching fluid simulation elements to surface geometry and topology," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 1–9, 2010.

[26] Bush, J., *MIT Lecture Notes on Surface Tension*. Massachusetts Institute of Technology, 2004.

[27] Campen, M. and Kobbelt, L., "Exact and robust (self-) intersections for polygonal meshes," in *Computer Graphics Forum*, vol. 29, pp. 397–406, John Wiley & Sons, 2010.

[28] Capell, S., Green, S., Curless, B., Duchamp, T., and Popović, Z., "Interactive skeleton-driven dynamic deformations," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 586–593, 2002.

[29] Capell, S., Green, S., Curless, B., Duchamp, T., and Popović, Z., "A multiresolution framework for dynamic deformations," in *Proc. Symposium on Computer Animation*, pp. 41–47, 2002.

[30] Carlson, M., Mucha, P. J., III, R. B. V. H., and Turk, G., "Melting and flowing," in *Proc. Symposium on Computer Animation*, pp. 167–174, July 2002.

[31] Chentanez, N., Feldman, B. E., Labelle, F., O'Brien, J. F., and Shewchuk, J. R., "Liquid simulation on lattice-based tetrahedral meshes," in *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, (Aire-la-Ville, Switzerland, Switzerland), pp. 219–228, Eurographics Association, 2007.

[32] Clanet, C. and Villermaux, E., "Life of a smooth liquid sheet," *Journal of Fluid Mechanics*, vol. 462, pp. 307–340, 2002.

[33] Clavet, S., Beaudoin, P., and Poulin, P., "Particle-based viscoelastic fluid simulation," in *Proc. Symposium on Computer Animation*, pp. 219–228, july 2005.

[34] Cleary, P. W., Pyo, S. H., Prakash, M., and Koo, B. K., "Bubbling and frothing liquids," *ACM Trans. Graph.*, vol. 26, no. 3, p. 97, 2007.

[35] Cohen, J. M. and Molemaker, M. J., "Practical simulation of surface tension flows," 2004. SIGGRAPH Sketch.

[36] De Gennes, P., Brochard-Wyart, F., Quéré, D., and Reisinger, A., *Capillarity and wetting phenomena: drops, bubbles, pearls, waves*. Springer Verlag, 2004.

[37] De Luca, L. and Costa, M., "Instability of a spatially developing liquid sheet," *Journal of Fluid Mechanics*, vol. 331, pp. 127–144, 2009.

[38] Deegan, R., Brunet, P., and Eggers, J., "Rayleigh-Plateau instability causes the crown splash," *Arxiv preprint arXiv:0806.3050*, 2008.

[39] Desbrun, M., Meyer, M., Schröder, P., and Barr, A., "Implicit fairing of irregular meshes using diffusion and curvature flow," *Proc. SIGGRAPH*, pp. 317–324, 1999.

[40] Desbrun, M., Schröder, P., and Wardetzky, M., "Discrete differential geometry: An applied introduction," in *SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses*, (New York, NY, USA), pp. 1–4, ACM, 2008.

[41] Dey, T., Edelsbrunner, H., Guha, S., and Nekhayev, D. V., "Topology preserving edge contraction," *Publ. Inst. Math. (Beograd) (N.S.)*, vol. 66, pp. 23–45, 1999.

[42] Du, J., Fix, B., Glimm, J., Jia, X., Li, X., Li, Y., and Wu, L., "A simple package for front tracking," *Journal of Computational Physics*, vol. 213, no. 2, pp. 613–628, 2006.

[43] Eckstein, I., Pons, J.-P., Tong, Y., Kuo, C.-C. J., and Desbrun, M., "Generalized surface flows for mesh processing," in *SGP '07: Proceedings of the Eurographics symposium on Geometry processing*, pp. 183–192, 2007.

[44] Edelsbrunner, H. and Harer, J., *Computational Topology, An Introduction*. American Mathematical Society, January 2010.

[45] Enright, D., Nguyen, D., Gibou, F., and Fedkiw, R., "Using the particle level set method and a second order accurate pressure boundary condition for free surface flows," in *Proc. 4th ASME-JSME Joint Fluids Eng. Conf., number FEDSM2003–45144. ASME*, Citeseer, 2003.

[46] Enright, D., Fedkiw, R., Ferziger, J., and Mitchell, I., "A hybrid particle level set method for improved interface capturing," *J. Comput. Phys.*, vol. 183, no. 1, pp. 83–116, 2002.

[47] Enright, D., Marschner, S., and Fedkiw, R., "Animation and rendering of complex water surfaces," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 736–744, 2002.

[48] Faloutsos, P., van de Panne, M., and Terzopoulos, D., "Dynamic free-form deformations for animation synthesis," *IEEE TVCG*, vol. 3, no. 3, pp. 201–214, 1997.

[49] Fedkiw, R., Stam, J., and Jensen, H. W., "Visual simulation of smoke," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 15–22, ACM, 2001.

[50] Foster, N. and Fedkiw, R., "Practical animation of liquids," in *SIGGRAPH '01*, (New York, NY, USA), pp. 23–30, ACM, 2001.

[51] Galoppo, N., Otaduy, M., Mecklenburg, P., Gross, M., and Lin, M., "Fast simulation of deformable models in contact using dynamic deformation textures," in *Proc. Symp. on Computer Animation*, pp. 73–82, 2006.

[52] Garland, M. and Heckbert, P., "Surface simplification using quadric error metrics," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 209–216, ACM Press/Addison-Wesley Publishing Co., 1997.

[53] Geiger, W., Leo, M., Rasmussen, N., Losasso, F., and Fedkiw, R., "So real it'll make you wet," in *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, (New York, NY, USA), p. 20, ACM, 2006.

[54] Glimm, J., Grove, J. W., and Li, X. L., "Three dimensional front tracking," *SIAM J. Sci. Comp*, vol. 19, pp. 703–727, 1998.

[55] Goktekin, T. G., Bargteil, A. W., and O'Brien, J. F., "A method for animating viscoelastic fluids," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 463–468, 2004.

[56] Greenwood, S. and House, D., "Better with bubbles: enhancing the visual realism of simulated fluid," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 287–296, Eurographics Association, 2004.

[57] Guendelman, E., Selle, A., Losasso, F., and Fedkiw, R., "Coupling water and smoke to thin deformable and rigid shells," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 973–981, 2005.

[58] Hirt, C. W. and Nichols, B. D., "Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries," *J. Comp. Phys.*, vol. 39, pp. 201–225, 1981.

[59] Hochstein, J. I. and Williams, T. L., "An implicit surface tension model," in *AIAA Meeting Papers*, vol. 96-0599, 1996.

[60] Hong, J.-M. and Kim, C.-H., "Discontinuous fluids," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 915–920, 2005.

[61] Hong, J., Lee, H., Yoon, J., and Kim, C., "Bubbles alive," in *ACM SIGGRAPH 2008 papers*, pp. 1–4, ACM, 2008.

[62] Hoppe, H., "Progressive meshes," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 99–108, ACM, 1996.

[63] Irving, G., Teran, J., and Fedkiw, R., "Invertible finite elements for robust simulation of large deformation," in *Proc. Symposium on Computer Animation*, pp. 131–140, 2004.

[64] Irving, G., Schroeder, C., and Fedkiw, R., "Volume conserving finite element simulations of deformable models," *ACM Trans. Graph.*, vol. 26, no. 3, p. 13, 2007.

[65] Ivanov, I., *Thin Liquid Films: Fundamentals and Applications*. CRC, 1988.

[66] Jiao, X., "Face offsetting: A unified approach for explicit moving interfaces," *J. Comput. Phys.*, vol. 220, no. 2, pp. 612–625, 2007.

[67] Ju, T., Losasso, F., Schaefer, S., and Warren, J., "Dual contouring of hermite data," in *ACM SIGGRAPH 2002 papers*, pp. 339–346, ACM New York, NY, USA, 2002.

[68] Kang, M., Fedkiw, R. P., and Liu, X.-D., "A boundary condition capturing method for multiphase incompressible flow," *J. Sci. Comput.*, vol. 15, no. 3, pp. 323–360, 2000.

[69] Kass, M. and Miller, G., "Rapid, stable fluid dynamics for computer graphics," in *SIGGRAPH '90*, (New York, NY, USA), pp. 49–57, ACM, 1990.

[70] Kass, M., Witkin, A., and Terzopoulos, D., "Snakes: active contour models," *Int. Journal Computer Vision*, vol. 1(4), pp. 321–331, 1988.

[71] Keiser, R., Adams, B., Gasser, D., Bazzi, P., Dutré, P., and Gross, M., "A unified Lagrangian approach to solid-fluid animation," in *the Proceedings of Eurographics Symposium on Point-based Graphics*, pp. 125–133, 2005.

[72] Kim, B., Liu, Y., Llamas, I., Jiao, X., and Rossignac, J., "Simulation of bubbles in foam with the volume control method," *ACM Trans. Graph.*, vol. 26, no. 3, p. 98, 2007.

[73] Kim, B., Liu, Y., Llamas, I., and Rossignac, J., "Advections with significantly reduced dissipation and diffusion," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 1, pp. 135–144, 2007.

[74] Kim, D., Song, O., and Ko, H., "A practical simulation of dispersed bubble flow," in *ACM SIGGRAPH 2010 papers*, pp. 1–5, ACM, 2010.

[75] Kim, D., Song, O.-y., and Ko, H.-S., "Stretching and wiggling liquids," in *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, (New York, NY, USA), pp. 1–7, ACM, 2009.

[76] Kim, T. and Carlson, M., "A simple boiling module," in *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 27–34, 2007.

[77] Kim, T., Thuerey, N., James, D., and Gross, M., "Wavelet turbulence for fluid simulation," *ACM Trans. Graph.*, vol. 27, no. 3, p. 50, 2008.

[78] Klingner, B. M., Feldman, B. E., Chentanez, N., and O'Brien, J. F., "Fluid animation with dynamic meshes," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 820–825, 2006.

[79] Kobbelt, L., Botsch, M., Schwanecke, U., and Seidel, H., "Feature sensitive surface extraction from volume data," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 57–66, ACM New York, NY, USA, 2001.

[80] Labelle, F. and Shewchuk, J. R., "Isosurface stuffing: fast tetrahedral meshes with good dihedral angles," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 57:1–57:10, 2007.

[81] Lachaud, J.-O. and Taton, B., "Deformable model with a complexity independent from image resolution," *Comput. Vis. Image Underst.*, vol. 99, no. 3, pp. 453–475, 2005.

[82] Lachaud, J. and Taton, B., "Deformable model with adaptive mesh and automated topology changes," in *Proc. 4th Int. Conference on 3D Digital Imaging and Modeling, Banff, Canada, IEEE*, pp. 12–19, 2003.

[83] Lien, S. and Kajiya., J. T., "A symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra," *IEEE CG&A*, vol. 4, pp. 35–41, October 1984.

[84] Lin, S. and Jiang, W., "Absolute and convective instability of a radially expanding liquid sheet," *Physics of Fluids*, vol. 15, p. 1745, 2003.

[85] Lin, S., Lian, Z., and Creighton, B., "Absolute and convective instability of a liquid sheet," *Journal of Fluid Mechanics*, vol. 220, pp. 673–689, 1990.

[86] Lindstrom, P. and Turk, G., "Evaluation of memoryless simplification," *IEEE TVCG*, vol. 5, no. 2, pp. 98–115, 1999.

[87] Liu, X.-D., Osher, S., and Chan, T., "Weighted essentially non-oscillatory schemes," *J. Comput. Phys.*, vol. 115, no. 1, pp. 200–212, 1994.

[88] Lorensen, W. E. and Cline, H. E., "Marching cubes: A high resolution 3d surface construction algorithm," in *SIGGRAPH '87*, (New York, NY, USA), pp. 163–169, ACM, 1987.

[89] Losasso, F., Gibou, F., and Fedkiw, R., "Simulating water and smoke with an octree data structure," in *Proceedings of ACM SIGGRAPH 2004*, pp. 457–462, ACM Press, Aug. 2004.

[90] Losasso, F., Shinar, T., Selle, A., and Fedkiw, R., "Multiple interacting liquids," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 812–819, 2006.

[91] McInerney, T. and Terzopoulos, D., "T-snakes: Topology adaptive snakes," *Medical Image Analysis*, vol. 4, no. 2, pp. 73–91, 2000.

[92] Meyer, M., Desbrun, M., Schröder, P., and Barr, A., "Discrete differential-geometry operators for triangulated 2-manifolds," *Visualization and mathematics*, vol. 3, no. 7, pp. 34–57, 2002.

[93] Mihalef, V., Metaxas, D., and Sussman, M., "Simulation of two-phase flow with sub-scale droplet and bubble effects," in *Proceedings of Eurographics 2009, CGF*, vol. 28:2, 2009.

[94] Molemaker, J., Cohen, J. M., Patel, S., and Noh, J., "Low viscosity flow simulations for animation," in *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (Aire-la-Ville, Switzerland, Switzerland), pp. 9–18, Eurographics Association, 2008.

[95] Molino, N., Bao, Z., and Fedkiw, R., "A virtual node algorithm for changing mesh topology during simulation," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 385–392, 2004.

[96] Molino, N., Bridson, R., Teran, J., and Fedkiw, R., "A crystalline, red green strategy for meshing highly deformable objects with tetrahedra," in *IMR*, pp. 103–114, 2003.

[97] Montani, C., Scateni, R., and Scopigno, R., "A modified look-up table for implicit disambiguation of marching cubes," *The Visual Computer*, vol. 10, no. 6, pp. 353–355, 1994.

[98] Müller, M., "Fast and robust tracking of fluid surfaces," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 237–245, ACM, 2009.

[99] Müller, M. and Gross, M., "Interactive virtual materials," in *the Proccedings of Graphics Interface*, pp. 239–246, 2004.

[100] Müller, M., Charypar, D., and Gross, M., "Particle-based fluid simulation for interactive applications," *Proc. of the ACM Siggraph/Eurographics Symposium on Computer Animation*, pp. 154–159, 2003.

[101] Müller, M., Dorsey, J., McMillan, L., Jagnow, R., and Cutler, B., "Stable real-time deformations," in *Proc. Symposium on Computer Animation*, pp. 49–54, 2002.

[102] Müller, M., Heidelberger, B., Teschner, M., and Gross, M., "Meshless deformations based on shape matching," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 471–478, 2005.

[103] Müller, M., Keiser, R., Nealen, A., Pauly, M., Gross, M., and Alexa, M., "Point based animation of elastic, plastic and melting objects," in *Proc. Symposium on Computer Animation*, 2004.

[104] Müller, M., Teschner, M., and Gross, M., "Physically-based simulation of objects represented by surface meshes," in *Computer Graphics International*, pp. 26–33, June 2004.

[105] Narain, R., Sewall, J., Carlson, M., and Lin, M. C., "Fast animation of turbulence using energy transport and procedural synthesis," in *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, (New York, NY, USA), pp. 1–8, ACM, 2008.

[106] Nesme, M., Kry, P., Jeřábková, L., and Faure, F., "Preserving topology and elasticity for embedded deformable models," in *ACM SIGGRAPH 2009 papers*, p. 52, ACM, 2009.

[107] Newmark, N. M., "A method of computation for structural dynamics," *ASCE J. Eng. Mech. Div.*, vol. 85, pp. 67–94, 1959.

[108] Nielson, G. and Hamann, B., "The asymptotic decider: resolving the ambiguity in marching cubes," in *Proceedings of the 2nd conference on Visualization'91*, pp. 83–91, IEEE Computer Society Press Los Alamitos, CA, USA, 1991.

[109] Nooruddin, F. S. and Turk, G., "Interior/exterior classification of polygonal models," in *VIS '00: Proceedings of the conference on Visualization '00*, (Los Alamitos, CA, USA), pp. 415–422, IEEE Computer Society Press, 2000.

[110] O'Brien, J. F., Bargteil, A. W., and Hodgins, J. K., "Graphical modeling and animation of ductile fracture," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 291–294, 2002.

[111] O'Brien, J. F. and Hodgins, J. K., "Graphical modeling and animation of brittle fracture," in *SIGGRAPH '99*, (New York, NY, USA), pp. 137–146, ACM Press/Addison-Wesley Publishing Co., 1999.

[112] Osher, S. and Fedkiw, R., *The Level Set Method and Dynamic Implicit Surfaces*. Springer-Verlag, New York, 2003.

[113] Osher, S. and Sethian, J., "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics*, vol. 79, pp. 12–49, 1988.

[114] Pauly, M., Keiser, R., Adams, B., Dutré;, P., Gross, M., and Guibas, L. J., "Meshless animation of fracturing solids," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 957–964, 2005.

[115] Peskin, C. S., "The immersed boundary method," *Acta Numerica*, vol. 11, pp. 1–39, 2002.

[116] Peskin, C. S., "Numerical analysis of blood flow in the heart," *Journal of Computational Physics*, vol. 25, pp. 220–252, November 1977.

[117] Pinkall, U. and Polthier, K., "Computing discrete minimal surfaces and their conjugates," *Experimental Mathematics*, vol. 2, no. 1, pp. 15–36, 1993.

[118] Pons, J.-P. and Boissonnat, J.-D., "Delaunay deformable models: Topology-adaptive meshes based on the restricted Delaunay triangulation," *Proceedings of CVPR '07*, pp. 1–8, 2007.

[119] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T., *Numerical Recipes in C*. Cambridge University Press, second ed., 1994.

[120] Rayleigh, L., "Investigation of the character of the equilibrium of an incompressible heavy fluid of variable density," *Proc. Lond. Math. Soc*, vol. 14, no. 1, pp. 170–177, 1883.

[121] Reynolds, C. W., "Adaptive polyhedral resampling for vertex flow animation, *unpublished*. http://www.red3d.com/cwr/papers/1992/df.html," 1992.

[122] Rivers, A. R. and James, D. L., "Fastlsm: fast lattice shape matching for robust real-time deformation," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 82:1–82:6, 2007.

[123] Rosenfeld, A., "Digital topology," *American Mathematical Monthly*, vol. 86, pp. 621–630, 1979.

[124] Schaefer, S. and Warren, J., "Dual Marching Cubes: primal contouring of dual grids," *Computer Graphics Forum*, vol. 24, no. 2, pp. 195–201, 2005.

[125] Schechter, H. and Bridson, R., "Evolving sub-grid turbulence for smoke animation," in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 1–7, Eurographics Association, 2008.

[126] Sederberg, T. W. and Parry, S. R., "Free-form deformation of solid geometric models," *SIGGRAPH Comput. Graph.*, vol. 20, no. 4, pp. 151–160, 1986.

[127] Selle, A., Fedkiw, R., Kim, B., Liu, Y., and Rossignac, J., "An unconditionally stable maccormack method," *J. Sci. Comput.*, vol. 35, no. 2-3, pp. 350–371, 2008.

[128] Serra, J., *Image analysis and mathematical morphology*. Academic Press, Inc. Orlando, FL, USA, 1983.

[129] Sethian, J. A., "A fast marching level set method for monotonically advancing fronts," *Proc. of the National Academy of Sciences of the USA*, vol. 93, pp. 1591–1595, February 1996.

[130] Shewchuk, J. R., "What is a good linear element? interpolation, conditioning, and quality measures," in 11^{th} *Int. Meshing Roundtable*, pp. 115–126, 2002.

[131] Sifakis, E., Der, K. G., and Fedkiw, R., "Arbitrary cutting of deformable tetrahedralized objects," in *Proc. Symposium on Computer Animation*, pp. 73–80, 2007.

[132] Sifakis, E., Shinar, T., Irving, G., and Fedkiw, R., "Hybrid simulation of deformable solids," in *Proc. Symposium on Computer Animation*, pp. 81–90, 2007.

[133] Sin, F., Bargteil, A., and Hodgins, J., "A point-based method for animating incompressible flow," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2009.

[134] Stam, J., "Stable fluids," in *SIGGRAPH '99*, (New York, NY, USA), pp. 121–128, ACM Press/Addison-Wesley Publishing Co., 1999.

[135] Strain, J. A., "A fast semi-lagrangian contouring method for moving interfaces," *Journal of Computational Physics*, vol. 169, pp. 1–22, May 2001.

[136] Sussman, M., "A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles," *J. Comp. Phys.*, vol. 187/1, 2003.

[137] Sussman, M. and Ohta, M., "A stable and efficient method for treating surface tension in incompressible two-phase flow," *SIAM Journal on Scientific Computing*, vol. 31, no. 4, pp. 2447–2471, 2009.

[138] Sussman, M. and Ohta, M., "A method for overcoming the surface tension time step constraint in multiphase flows II," *International Journal for Numerical Methods in Fluid (submitted)*, 2010.

[139] Taubin, G., "A signal processing approach to fair surface design," *Computer Graphics*, vol. 29, no. Annual Conference Series, pp. 351–358, 1995.

[140] Taylor, G., "The instability of liquid surfaces when accelerated in a direction perpendicular to their planes. I," *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 201, no. 1065, pp. 192–196, 1950.

[141] Teran, J., Sifakis, E., Blemker, S., Ng-Thow-Hing, V., Lau, C., and Fedkiw, R., "Creating and simulating skeletal muscle from the visible human data set," *IEEE Transactions on Visualization and Computer Graphics*, pp. 317–328, 2005.

[142] Terzopoulos, D. and Fleischer, K., "Deformable models," *The Visual Computer*, vol. 4, pp. 306–331, 1988.

[143] Terzopoulos, D. and Fleischer, K., "Modeling inelastic deformation: Viscoelasticity, plasticity, fracture," in *the Proceedings of ACM SIGGRAPH 1988*, pp. 269–278, Aug. 1988.

[144] Terzopoulos, D., Platt, J., and Fleischer, K., "Heating and melting deformable models (from goop to glop)," in *the Proceedings of Graphics Interface*, pp. 219–226, June 1989.

[145] Thürey, N., Wojtan, C., Gross, M., and Turk, G., "A multiscale approach to mesh-based surface tension flows," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 1–10, 2010.

[146] Tournois, J., Wormser, C., Alliez, P., and Desbrun, M., "Interleaving delaunay refinement and optimization for practical isotropic tetrahedron mesh generation," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 1–9, 2009.

[147] Treuille, A., Lewis, A., and Popović, Z., "Model reduction for real-time fluids," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 826–834, 2006.

[148] Turk, G., "Generating textures on arbitrary surfaces using reaction-diffusion," in *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pp. 289–298, ACM, 1991.

[149] Varadhan, G., Krishnan, S., Sriram, T., and Manocha, D., "Topology preserving surface extraction using adaptive subdivision," in *Proceedings of SGP '04*, pp. 235–244, ACM, 2004.

[150] Wang, H., Miller, G., and Turk, G., "Solving general shallow wave equations on surfaces," in *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, (Aire-la-Ville, Switzerland, Switzerland), pp. 229–238, Eurographics Association, 2007.

[151] Wang, H., Mucha, P. J., and Turk, G., "Water drops on surfaces," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 921–929, 2005.

[152] Wardetzky, M., Mathur, S., Kälberer, F., and Grinspun, E., "Discrete laplace operators: no free lunch," in *Proceedings of the fifth Eurographics symposium on Geometry processing*, p. 37, Eurographics Association, 2007.

[153] Whited, B. and Rossignac, J., "Relative blending," *Computer-Aided Design*, vol. 41, no. 6, pp. 456–462, 2009.

[154] Wojtan, C., Thürey, N., Gross, M., and Turk, G., "Deforming meshes that split and merge," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 1–10, 2009.

[155] Wojtan, C., Thürey, N., Gross, M., and Turk, G., "Physics-inspired topology changes for thin fluid features," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 1–8, 2010.

[156] Wojtan, C. and Turk, G., "Fast viscoelastic behavior with thin features," *ACM Trans. Graph.*, vol. 27, no. 3, p. 47, 2008.

[157] Zaharescu, A., Boyer, E., and Horaud, R., "Transformesh: a topology-adaptive mesh-based approach to surface evolution," *Lecture Notes in Computer Science*, vol. 4844, p. 166, 2007.

[158] Zheng, W., Yong, J.-H., and Paul, J.-C., "Simulation of bubbles," in *Proc. of the ACM Siggraph/Eurographics Symposium on Computer Animation*, pp. 325–333, 2006.

[159] Zhu, Y. and Bridson, R., "Animating sand as a fluid," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 965–972, 2005.

[160] Zorin, D., Schröder, P., and Sweldens, W., "Interpolating subdivision for meshes with arbitrary topology," in *ACM SIGGRAPH 1996 papers*, p. 192, ACM, 1996.

INDEX

INDEX

VITA

Chris Wojtan grew up in Homer Glen, Illinois, USA. He graduated from Lockport Township High School in Lockport, Illinois in 2000. Chris attended the University of Illinois in Urbana-Champaign (UIUC), where he majored in Computer Science and minored in Mathematics and Physics. As an undergraduate at UIUC, Chris performed research with Professors Michael Garland and Yizhou Yu. Chris graduated with Honors on the Dean's List from UIUC with the title of James Scholar in 2004.

In the summer of 2004, Chris Wojtan worked at Lawrence Livermore National Laboratory (LLNL) in Livermore, California, USA as a summer scholar. At LLNL, Chris contributed to the R&D 100 Award-winning software VisIt.

Chris Wojtan attended the Georgia Institute of Technology (Georgia Tech) for his graduate studies, under the guidance of Professor Greg Turk. In 2006, Chris worked at Carnegie Mellon University in Pittsburgh, Pennsylvania, USA, where he collaborated with Professors Adam Bargteil and Jessica K. Hodgins. Periodically throughout 2008 and 2009, Chris worked at ETH Zürich in Zürich, Switzerland, where he collaborated with Professor Markus Gross and Doctor Nils Thürey. Chris was awarded a Presidential Fellowship from Georgia Tech in 2004, a National Science Foundation Graduate Fellowship in 2005, and an Outstanding Graduate Research Assistant Award by the Georgia Tech College of Computing in 2010.